CA ERwin® Process Modeler

API Reference Guide

r7.3

CA Product References

This document references the following CA products:

- CA ERwin® Process Modeler (CA ERwin PM)
- CA ERwin® Data Modeler (CA ERwin DM)
- CA ERwin® Model Manager (CA ERwin MM)

Contact CA

Contact Technical Support

For your convenience, CA provides one site where you can access the information you need for your Home Office, Small Business, and Enterprise CA products. At http://ca.com/support, you can access:

- Online and telephone contact information for technical assistance and customer services
- Information about user communities and forums
- Product and documentation downloads
- CA Support policies and guidelines
- Other helpful resources appropriate for your product

Provide Feedback

If you have comments or questions about CA product documentation, you can send a message to techpubs@ca.com.

If you would like to provide feedback about CA product documentation, please complete our short customer survey, which is also available on the CA support website, found at http://ca.com/support.

Contents

Chapter 1: Introduction

The CA ERwin PM Script Client API (CA ERwin PM API) provides advanced customization capabilities that enable you to access and manipulate modeling data in CA ERwin PM memory at runtime, as well as in models persisted in files. The CA ERwin PM API interfaces are automation-compatible and provide extensive design and runtime facilities for third party integrators as well as users of script-based environments.

The CA ERwin PM API enables you to complement features in CA ERwin PM with custom components. You can implement custom components using scripts, add-ins, and COM-based API technologies. Because of its flexibility the CA ERwin PM API makes it possible for you to seamlessly integrate CA ERwin PM modeling features in your client development cycle.

This section contains the following topics:

Major Features

The CA ERwin PM API is a group of interfaces that include the following functionality:

Active Model Data Objects (AMDO)

Allows a third-party client to access model data through a COM automation-compatible API. This feature is the major component in the API functionality. All interfaces that comprise the CA ERwin PM API are automation-based, and are therefore dual. These dual interfaces allow you faster access to methods and properties. Using dual interfaces, you can directly call the functions without using an *Invoke()* function.

Collections and enumerators

Facilitates programming constructions in script languages that target the AMDO automation features.

Connection points

Supports the sync event facilities of languages. The CA ERwin PM API delivers a collection of connection points interfaces and support for the *ITypeInfo2* interface.

IErrorInfo interfaces

Provide automation-rich error handling when they are exposed by the CA ERwin PM API components.

Active Scripting

Hosts a scripting environment and provides an invocation mechanism for script and add-in components. A mechanism is provided to register add-ins and scriplets with the Active Scripting environment.

Typical Use Cases

The CA ERwin PM API provides a wide range of integration solutions for using CA ERwin PM functionality with complex business processes.

Many users of the CA ERwin PM API are automation and script-based clients. These clients have very specific interface design requirements imposed by COM automation standards. For instance, they are often limited to a single incoming and outgoing interface exposed by any particular COM object. This limitation is due to the fact that the only recognizable interface type for pure automation is *IDispatch* and it renders the use of *QueryInterface* functionality unfit. The common technique to address the problem includes *Alternate Identities* and read-only properties that expose secondary interfaces.

Another example of a targeted domain customer is one using alternative (not C++) languages to implement a client. The list includes Visual Basic, VB Script, Java Script, and so on. The list includes specially tailored language idioms to encapsulate language-COM binding, such as collections of objects, connection points, rich error handling, and so on.

The CA ERwin PM API combines a number of components and presents them as a set of interfaces accessible using COM. For information that describes the CA ERwin PM API components, see the chapter API Components (see page 13).

Standalone Client

The CA ERwin PM API can be used within a standalone executable. In this mode, the third-party customer activates the API as an in-process server. For more information on how to use this functionality, see the section The API as a Standalone Executable (see page 27).

Add-in Component

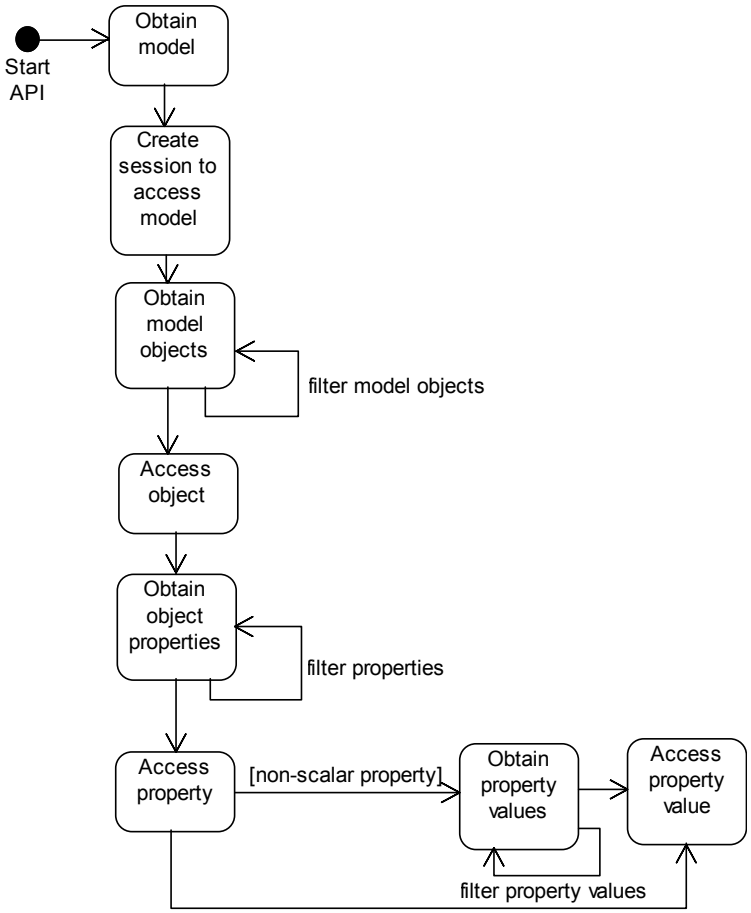
CA ERwin PM provides the capability to host third-party add-in modules. In this mode, the API client is executed within the CA ERwin PM application environment. Changes made by the API client can be observed in the CA ERwin PM user interface. For more information on how to use this functionality, see the section The API as an Add-in Tool (see page 23).

Chapter 2: API Components

This section contains the following topics:

Overview

The API is a collection of COM interfaces. The main activities that are performed using the API are shown in the following diagram:



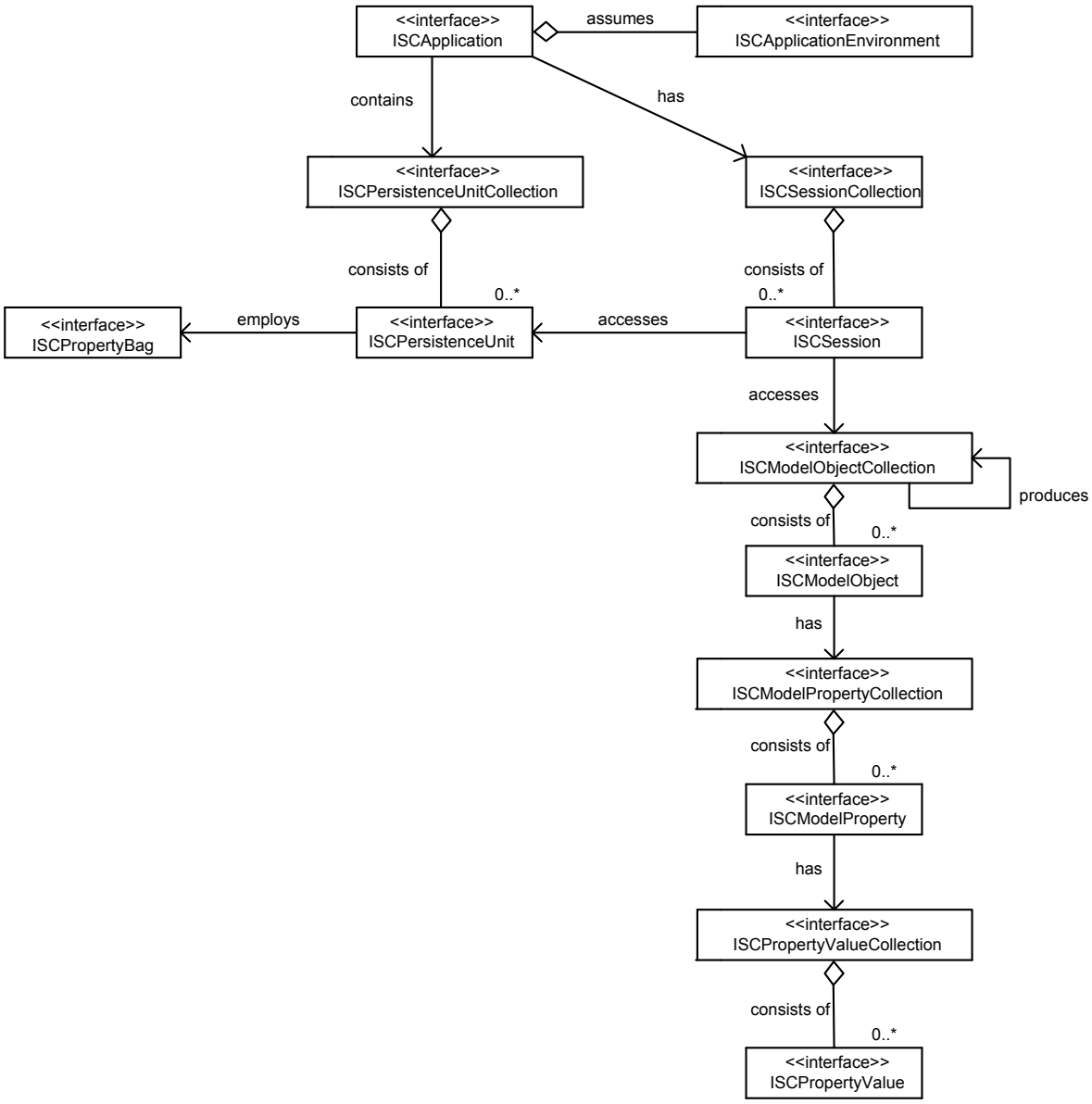
The following table summarizes the interfaces and their descriptions for each API activity:

Activity	Interfaces	Description
Start API	ISCApplication ISCApplicationEnvironment	This is the entry point for the API. *ISCApplication* holds a list of available models (persistence units), and connections (sessions) between the API client and the models. *ISCApplicationEnvironment* contains the information about the runtime environment.
Obtain model	ISCPersistenceUnit ISCPersistenceUnitCollection	In the API, models are represented as persistence units. An *ISCPersistenceUnit* is an active model within CA ERwin PM. *ISCPersistenceUnitCollection* is a collection of persistence units and is maintained by the *ISCApplication* interface.
Create session to access model	ISCSession ISCSessionCollection	An *ISCSession* is an active connection between the API client and a model. In order to access models, API clients create sessions and open them against persistence units. *ISCSessionCollection* is a collection of sessions and is maintained by the *ISCApplication* interface.
Obtain model objects	ISCModelObjectCollection	*ISCModelObjectCollection* is a collection of objects in the model that is connected to the active session. Membership in this collection can be limited by establishing filter criteria.
Access object	ISCModelObject	*ISCModelObject* represents an object in a model (for example, activity, data store, cost center, and so on).

Activity	Interfaces	Description
Obtain object properties	ISCModelPropertyCollection	*ISCModelPropertyCollection* is a collection of properties for a given model object. Membership in this collection can be limited by establishing filter criteria.
Access property	ISCModelProperty	*ISCModelProperty* represents a property of a given object.
Obtain property values	ISCPropertyValueCollection	*ISCPropertyValueCollection* is used to obtain property values, when certain properties in CA ERwin PM can contain more than one value (for example, if the properties are non-scalar).
Access property value	ISCPropertyValue	*ISCPropertyValue* is a single value of a given property.

Relationships of the API Interfaces

The following shows the relationships of the CA ERwin PM API:



Object Identifiers

The CA ERwin PM API presents data in object/property form. For example, in a CA ERwin PM model an activity is represented by an instance of a *PMActivity* object. The name of the activity is contained in the name property of the *PMActivity* object.

Each object must bear an identifier-a value that uniquely identifies the object instance. Internally, object identifiers are 20 bytes long. They contain two components: a GUID (also known as a UUID) in the first 16 bytes, and a 32-bit unsigned integer suffix in the last 4 bytes.

A GUID contains the following components:

- One 32-bit unsigned integer
- Two 16-bit unsigned integers
- Eight 8-bit unsigned integers (represented as unsigned characters)

These components total 128 bits, or 16 bytes. Therefore, an object identifier contains an extra 32-bit unsigned integer (the 4 byte suffix) at the end for a total of 160 bits, or 20 bytes.

To simplify working with object identifiers, and because of COM automation limitations on datatypes, the API uses a string to represent identifiers.

The following table lists the aliases used in this guide and in the interface definitions:

Type Name	Format	Use
SC_OBJID	{xxxxxxxx-xxxx-xxxx-xxxx-xxxxxxxxxxxx}+suffix	Object identifier
SC_CLSID	{xxxxxxxx-xxxx-xxxx-xxxx-xxxxxxxxxxxx}+suffix	Class (object, property type, and so on) identifier
SC_MODELTYPEID	{xxxxxxxx-xxxx-xxxx-xxxx-xxxxxxxxxxxx}+suffix	Model type identifier
SC_CREATORID	{xxxxxxxx-xxxx-xxxx-xxxx-xxxxxxxxxxxx}	Creator identifier

One set of object identifiers is predefined-those identifiers whose GUID component contains zero. If the final 4 bytes of the identifier also contain zero, the identifier represents a null identifier. Other values of the offset are reserved for future use.

Object Types and Property Types Codes

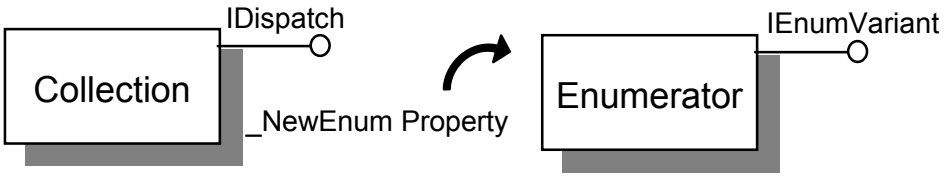
The identifiers for object and property types in CA ERwin PM are all longs and not GUIDs. The API uses GUIDs for all identifiers. To handle this situation, the following scheme is used. The first set of data in the GUID is the hexadecimal equivalent of the identifier for the object or property type. The rest of the fields in the GUID are zeros. For example, if the GUID for an object or a property type is:

{000003E8-0000-0000-0000-000000000000}+00000000

The type identifier is 0x3E8 or 1000, which is a *PMDiagram*.

Collections and Automation

Automation defines the *IEnumVARIANT* interface to provide a standard method for the API clients to iterate over collections. Every collection interface in the API exposes a read-only property named *_NewEnum* to let the API clients know that the collection supports iteration. The *_NewEnum* property returns a pointer on the *IEnumVARIANT* interface.



The *IEnumVARIANT* interface provides a way to iterate through the items contained by a collection. This interface is supported by an enumerator interface that is returned by the *_NewEnum* property of the collection, as shown in the above illustration.

The *IEnumVARIANT* interface defines the following member functions:

Next

Retrieves one or more elements in a collection, starting with the current element.

Skip

Skips over one or more elements in a collection.

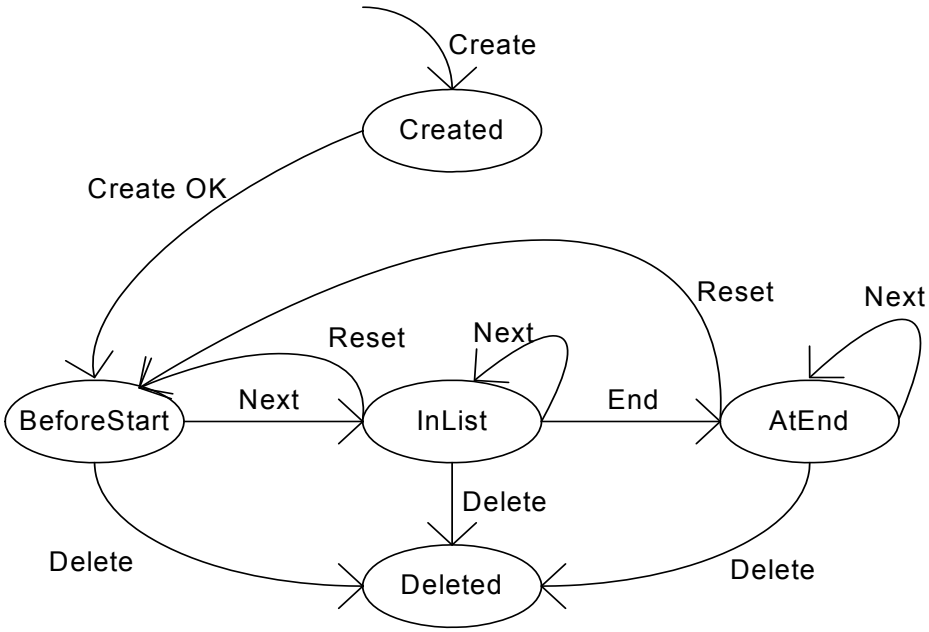
Reset

Resets the current element to the first element in the collection.

Clone

Copies the current state of the enumeration so you can return to the current element after using Skip or Reset.

The *IEnumVARIANT* collection implements a Rogue Wave Software, Inc. style *advance and return* iteration. For this reason, the iterator has the following life cycle:



IPtUCModelPropIter Life Cycle

When the iterator is created, it enters the *Created* state. It then forces itself into the *BeforeStart* state. A successful advance drives the iterator into the *InList* state, while an unsuccessful advance drives it into the *AtEnd* state. A *Reset* drives the iterator back to the *BeforeStart* state, and deletion drives it into the *Deleted* state.

The iterator is positioned over a member of the collection (that is, is associated with a current member) if and only if it is in the *InList* state.

NewEnum Property of a Collection Object

The *_NewEnum* property identifies support for iteration through the *IEnumVARIANT* interface. The *_NewEnum* property has the following requirements:

- Named *_NewEnum*
- Returns a pointer to the enumerator *IUnknown* interface
- Dispatch identification for the property is:

  DISPID = DISPID_NEWENUM (-4)

Unused Interfaces

Both CA ERwin Data Modeler and CA ERwin PM use the same interfaces for their respective APIs. However, there are certain interfaces that are not used in the CA ERwin PM API. The behavior of such interfaces is undefined. The following interfaces are not used in the CA ERwin PM API:

- *ISCApplicationServiceCollection*
- *ISCApplicationService*
- *ISCApplicationWindow*
- *ISCApplicationWindowCollection*
- *ISCModelDirectoryCollection*
- *ISCModelDirectory*
- *ISCModelDirectoryUnit*
- *ISCModelSet*
- *ISCModelSetCollection*

Chapter 3: API Tasks

This chapter describes how to perform basic tasks using the CA ERwin PM API. Each task is documented with a table that lists the interfaces and methods needed for that task. In most cases, the table shows a subset of all the methods for the given interface. For a complete list of API interfaces and their respective methods, see the appendix API Interfaces Reference (see page 79).

This section contains the following topics:

Required File

The API is packaged as a COM Dynamic Link Library (DLL) named *PM_SCAPI.dll*. When CA ERwin PM is installed, *PM_SCAPI.dll* is copied to the Binaries directory (a subdirectory of the install directory). To use the API, the *PM_SCAPI.dll* file must be imported.

ISCApplication Object

The entry point into the interface hierarchy of the API is through the *ISCApplication* interface. The *ISCApplication* interface provides access to the persistence units and sessions. You must create an instance of *ISCApplication* prior to using any of the other interfaces in the API.

The following examples illustrate how to create the *ISCApplication* object:

Example: ISCApplication Object in C++

```
#import "PM_SCAPI.dll" using namespace SCAPI;

ISCApplicationPtr m_scAppPtr;
 HRESULT hr;

hr = m_scAppPtr.CreateInstance(__uuidof(SCAPI::Application));
```

Example: ISCApplication Object in Visual Basic

```
Dim m_scAppPtr As SCAPI.Application
Set m_scAppPtr = CreateObject("ProcessModeler.SCAPI")
```

Application Properties

You can get information about the CA ERwin PM application by using the following tables.

ISCApplication Interface

The following table contains information on the *ISCApplication* interface:

Signature	Description	Valid Arguments
BSTR Name()	Modeling Application Title	None
BSTR Version()	Modeling Application Version	None
BSTR ApiVersion()	API version	None
ISCApplicationEnvironment ApplicationEnvironment()	Reports attributes of runtime environment and available features such as add-in mode, user interface visibility, and so on	None
ISCSessionCollection *Sessions()	Returns a collection of sessions created within the application	None

ISCApplicationEnvironment

The following table contains information on the *ISCApplication* environment:

Signature	Description	Valid Arguments
ISCPropertyBag PropertyBag(VARIANT Category[optional], VARIANT Name[optional], VARIANT AsString[optional])	Populates a property bag with one or more property values as indicated by Category and Name	Category: ■ Empty – Complete set of features from all categories returned ■ VT_BSTR – Features returned from the given category Name: ■ Empty – All properties from the selected category are returned ■ VT_BSTR – The property with the given name and category returned AsString: ■ Empty – All values in the property bag are presented in their native type ■ VT_BOOL – If set to TRUE, all values in the property bag are presented as strings

Accessing a Model

An API client can either be a standalone executable that runs outside the CA ERwin PM environment or a DLL that is invoked from within the CA ERwin PM environment by clicking Add-Ins on the Tools menu.

The API as an Add-in Tool

When the API client is a DLL that is invoked by clicking Add-Ins on the CA ERwin PM Tools menu, the client runs within the environment of CA ERwin PM. As a result, all the models that are currently open within CA ERwin PM are populated in the *PersistenceUnits* property of the *ISCApplication* interface, when an instance of the interface is created.

To iterate through the models that are currently open in CA ERwin PM, you can use the *ISCApplication* interface, *ISCPersistenceUnitCollection* interface, and the *ISCPersistenceUnit* interface, which are described in the following sections.

ISCApplication Interface

The following table contains information on the *ISCApplication* interface:

Signature	Description	Valid Arguments
ISCPersistenceUnitCollection PersistenceUnits()	Returns a collection of all persistence units loaded in the application	None

ISCPersistenceUnitCollection Interface

The following table contains information on the *ISCPersistenceUnitCollection* interface:

Signature	Description	Valid Arguments
ISCPersistenceUnit Item(VARIANT nIndex)	Passes back a pointer for the *PersistenceUnit* component identified by its ordered position	nIndex: ■ VT_UNKNOWN – A pointer to a session. Retrieves the persistence unit associated with the session. ■ VT_I4 – Index within the collection. Collection index is from 0 to size-1. Retrieves the persistence unit in the collection with the given index.
long Count()	Number of persistence units in the collection	None

ISCPersistenceUnit Interface

The following table contains information on the *ISCPersistenceUnit* interface:

Signature	Description	Valid Arguments
BSTR Name()	Returns the name of the persistence unit	None
SC_OBJID ObjectId()	Returns an object identifier for the persistence unit	None

Signature	Description	Valid Arguments
ISCPropertyBag PropertyBag(VARIANT List[optional], VARIANT AsString[optional])	Returns a property bag with the properties of the persistence unit	List: ■ VT_BSTR – Semicolon-separated list of property names. Returns a property bag with the unit properties in the given list. AsString: ■ VT_BOOL – Returns a property bag with all values presented as strings if set to TRUE. Otherwise, the values are presented in its native format.
VARIANT_BOOL HasSession()	Returns TRUE if a unit has one or more sessions connected	None
VARIANT_BOOL IsValid()	Returns TRUE if self is valid	None

Property Bag Members for a Persistence Unit

The following table shows property names and descriptions for property bag members for a persistence unit:

Property Name	Type	Read-only	Description
Locator	BSTR	TRUE	Full path of the file that stores the model
Active Model	Boolean	TRUE	TRUE for the model of which the topmost diagram in the CA ERwin PM application belongs

ISCPropertyBag Interface

The following table contains information on the *ISCPropertyBag* interface:

Signature	Description	Valid Arguments
long Count()	Returns the number of properties	None
VARIANT Value(VARIANT Property)	Retrieves the indicated property name in the bag	Property: ■ VT_BSTR – Name of property. Value of the property with the given name in the property bag.
BSTR Name(long PropertyIdx)	Retrieves the indicated property name with the given index. Range of indices is from 0 to size-1.	None

The following examples illustrate how to use the API as an add-in tool to iterate through the open models:

Example: The API as an Add-in Tool Using C++

```
void APIExample::IteratePersistenceUnits(ISCApplicationPtr m_scAppPtr)
{
    ISCPersistenceUnitCollectionPtr m_scPUnitColPtr;
    m_scPUnitColPtr = m_scAppPtr->GetPersistenceUnits();

    ISCPersistenceUnitPtr pxSCPUnit = 0;
    long lCnt = m_scPUnitColPtr->GetCount();

    for(long i = 0; i < lCnt; i++)
    {
       pxSCPUnit = m_scPUnitColPtr->GetItem(i);
       CString csName = (LPSTR) pxSCPUnit->GetName();   // name of model
       ISCPropertyBagPtr pxPropBag = pxSCPUnit->GetPropertyBag
                ("Locator;Active Model");
       long index = 0;
       CComVariant vPathName = pxPropBag->GetValue(ColeVariant(index));  //
       full path of model
       index = 1;
       CComVariant cActiveModel = pxPropBag->GetValue(COleVariant(index)); // true if active model
       // …
    }
}
```

Example: The API as an Add-in Tool Using Visual Basic

```
Public Sub IteratePersistenceUnits(m_scAppPtr as        SCAPI.Application)

    Dim m_scPersistenceUnitCol as SCAPI.PersistenceUnits

    Dim numUnits As Integer
    Dim persUnit As SCAPI.PersistenceUnit

    Set m_scPersistenceUnitCol = m_scAppPtr.PersistenceUnits

    ' Load open units
    numUnits = m_scPersistenceUnitCol.Count
    If (numUnits > 0) Then
        For Each persUnit In m_scPersistenceUnitCol
            Dim propBag As SCAPI.PropertyBag
            Set propBag = persUnit.PropertyBag("Locator")
            Debug.Print persUnit.Name    ' name of model
            Debug.Print propBag.Value(0) ' full path of model
            ' …
        Next
    End If
End Sub
```

The API as a Standalone Executable

When the API client is a standalone executable, the client runs outside the CA ERwin PM environment. As a result, when the *ISCApplication* interface is created, the *PersistenceUnits* property is an empty collection. Even if CA ERwin PM is running and there are open models, the *PersistenceUnits* property is still empty because the API environment is independent of the CA ERwin PM environment. To get a valid persistence unit, the API client needs to either create a new model or open an existing model.

Creating a Model

To create a new model using the API, you first need to create a new instance of *ISCPropertyBag*. The *ISCPropertyBag* interface is a property bag that is used to hold the properties of the new model. The following properties are used in creating a new model:

Property Name	Type	Purpose
Name	BSTR	Name of the new model.
ModelType	VT_I2 or BSTR	Type of model. This determines what type of context activity to create. The valid values for *ModelType* are: ■ 0 – IDEF0 (this is the default if none provided) ■ 1 – DFD ■ 2 – IDEF3
UseBlankModel	Boolean	If set to TRUE, a new CA ERwin PM model is created with no default objects (such as fonts, colors, and so on). If set to FALSE, a CA ERwin PM model with default objects is created. If *UseBlankModel* is not explicitly defined in the property bag, a model with default objects is created.

Once the property bag is created and populated, a new persistence unit must be created within the persistence unit collection. The resulting persistence unit is a new model with default properties.

ISCPersistenceUnitCollection Interface

The following table contains information on the *ISCPersistenceUnitCollection* interface:

Signature	Description	Valid Arguments
ISCPersistenceUnit * Create(ISCPropertyBag * PropertyBag, VARIANT ObjectId [optional])	Creates a new unit, and registers the unit with the collection	ObjectId: ■ Empty – The CA ERwin PM API assigns an ID to the new persistence unit. ■ VT_BSTR – Object ID for the new persistence unit. The CA ERwin PM API assigns the given ID to the new persistence unit.

ISCPropertyBag Interface

The following table contains information on the *ISCPropertyBag* interface:

Signature	Description	Valid Arguments
VARIANT_BOOL Add(BSTR Name, VARIANT Value)	Adds a new property to the bag	Value: ■ All VARIANTs are valid. The function returns TRUE if the property was added to the bag, otherwise, it is FALSE.

The following examples illustrate how to create a new persistence unit:

Example: New Persistence Unit Creation Using C++

```
void APIExample::CreatePersistenceUnit(ISCPersistenceUnitCollectionPtr m_scPUnitColPtr)
{
   ISCPropertyBagPtr propBag;
   HRESULT hr =propBag.CreateInstance(__uuidof(SCAPI::PropertyBag));
   if (FAILED(hr))
     return;
   propBag->Add("Name", “Test Model”);
   propBag->Add("ModelType", “IDEF0”);
   ISCPersistenceUnitPtr m_scPUnitPtr = m_scPUnitColPtr->Create(propBag,vtMissing);
   // …
}
```

Example: New Persistence Unit Creation Using Visual Basic

```
Public Sub CreateNewModel(PersUnitCol As SCAPI.PersistenceUnits)
    Dim propBag As New SCAPI.PropertyBag
    Dim scPersistenceUnit As SCAPI.PersistenceUnit

    bRetVal = propBag.Add("Name", "Test Model")
    bRetVal = propBag.Add("ModelType", 0)
    Set scPersistenceUnit = PersUnitCol.Create(propBag)
End Sub
```

Opening an Existing Model

An existing CA ERwin PM model is opened by adding a persistence unit to the persistence unit collection (*ISCPersistenceUnitCollection*). When the API client is an add-in tool, opening a model through the API also opens the model in the CA ERwin PM user interface.

ISCPersistenceUnitCollection Interface

The following table contains information on the *ISCPersistenceUnitCollection* interface:

Signature	Description	Valid Arguments
ISCPersistenceUnit * Add(VARIANT Locator, VARIANT Disposition [optional])	Adds a new persistence unit to the unit collection	Locator: ■ VT_BSTR – Full path to the CA ERwin PM model. This is the model that is loaded into the persistence unit. Disposition: ■ Empty – This parameter is not used.

The following examples illustrate how to open an existing model:

Example: Open an Existing Model in C++

```
void APIExample::OpenModel(ISCPersistenceUnitCollectionPtr m_scPUnitColPtr, CString csFullPath)
{
    ISCPersistenceUnitPtr m_scPUnitPtr = m_scPUnitColPtr->Add(COleVariant(csFullPath));
    // …
}
```

Example: Open an Existing Model in Visual Basic

```
Public Function OpenModel(persUnitCol as SCAPI.PersistenceUnits, fullModelPath as String) _
   As SCAPI.PersistenceUnit

   Set OpenModel = persUnitCol.Add(fullModelPath)
End Sub
```

Opening a Session

Before the objects of a model can be accessed using the API, an *ISCSession* instance must first be established for the *ISCPersistenceUnit* of the model. To open a session for a persistence unit, add a new *ISCSession* to the *ISCSessionCollection*, and then open the *ISCPersistenceUnit* in the new session.

ISCSessionCollection Interface

The following table contains information on the *ISCSessionCollection* interface:

Signature	Description	Valid Arguments
ISCSession * Add()	Constructs a new, closed Session object, and adds it to the collection	None

ISCSession Interface

The following table contains information on the *ISCSession* interface:

Signature	Description	Valid Arguments
VARIANT_BOOL Open(ISCPersistenceUnit * Unit, VARIANT Level [optional], VARIANT Flags [optional])	Binds self to the persistence unit identified by the *Unit* parameter	Unit: ■ Pointer to a persistence unit that was loaded. Attaches the persistence unit to the session. Level: ■ Empty – Defaults to data level access (SCD_SL_M0) ■ SCD_SL_M0 – Data level access Flags: ■ Empty – Defaults to SCD_SF_NONE ■ SCD_SF_NONE – Specifies that other sessions can have access to the attached persistence unit ■ SCD_SF_EXCLUSIVE – Specifies that other sessions cannot have access to the attached persistence unit

The following examples illustrate how to open a session:

Example: Open a Session in C++

```
void APIExample::OpenSession(ISCApplicationPtr m_scApplicationPtr,
                             ISCPersistenceUnitPtr m_scPUnitPtr)
{
   ISCSessionCollectionPtr m_scSessionColPtr = m_scApplicationPtr->GetSessions();
   ISCSessionPtr m_scSessionPtr = m_scSessionColPtr->Add();  // add a new session
   CComVariant varResult = m_scSessionPtr->Open(m_scPUnitPtr, (long) SCD_SL_M0); // open unit
   if (varResult.vt == VT_BOOL && varResult.boolVal == FALSE)
     return;
   // …
}
```

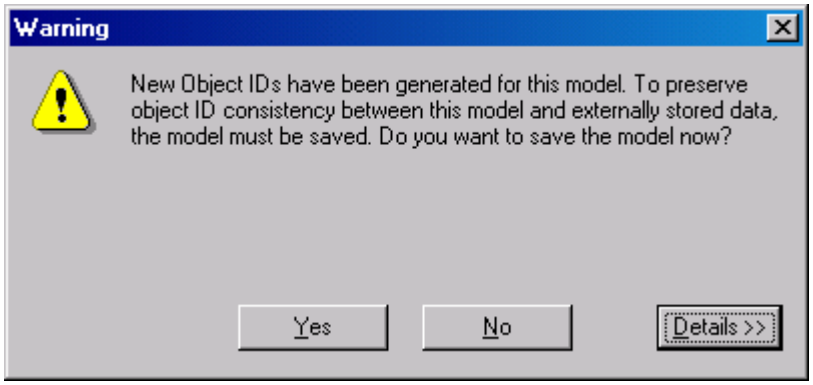
Example: Open a Session in Visual Basic

```
Public Sub OpenSession(scApp As SCAPI.Application, scPersistenceUnit As SCAPI.PersistenceUnit)
    Dim m_scSessionCol as SCAPI.Sessions
    Set m_scSessionCol = scApp.Sessions

      Set scSession = m_scSessionCol.Add  'new session
      bRetVal = scSession.Open(scPersistenceUnit, SCD_SL_M0)  ' open the
    persistence
 unit
End Sub
```

Preservation of Object IDs When You Open a Model

When an existing model is opened using the API, the API client may display the following message:



The object IDs that are used to identify the objects in a model from an external source are only generated when it is needed. In the context of the API, the *ISCModelObject:: ObjectId()* method returns this generated object ID. To persist these object IDs, the model must be saved. If the object IDs are not saved, a new set of IDs will be generated the next time the API client is executed. Examples of when the model should be saved to persist the generated object IDs include:

- The API client stores the object ID externally in a file or in a database.
- The API client needs the capability to access a specific object using its object ID.

There are scenarios where persisting the generated IDs is not necessary. Some examples include:

- The API client does not store any model data externally.
- Prior to running the API client, a lot of changes were made to the model. When you are prompted to save the model to preserve the object IDs, the model is saved in its entirety if you click *Yes*. If the changes that you made to the model are only temporary, the object IDs should not be preserved so you can avoid saving unwanted changes.

Alternative Procedure to Create a Model

You can use the API to create a default model and then set all the *PMModel* properties. However, it may be easier to create a standard model with all the model properties set the way you want, and then use it as a starting point for the creation of new models. In addition, standard UDPs can also be defined in this model, as well as default text fonts and colors.

The following describes how you can create a model:

- Open the product and create a new model.
- Click Model Properties from the Model menu to define general properties for the new model.
- Click Preferences from the Tools menu to define any additional default properties.
- Create standard UDP definitions and keywords.
- Create standard Bitmaps, Role Groups, and Roles.
- Make default shape assignments.
- Save the model and exit the product.
- Set the file attribute of the model to read-only.

Once the template model is created, you can create a new model by opening the template model through the API and changing the *PMFilename* property of the *PMModel* object. The following examples illustrate how to open a template model:

Example: Open a Template Model in C++

```
void APIExample::OpenTemplateModel(ISCPersistenceUnitCollectionPtr m_scPUnitColPtr,
                                   ISCSessionCollectionPtr m_scSessionColPtr, CString csFullPath)
{
  ISCPersistenceUnitPtr m_scPUnitPtr = m_scPUnitColPtr->Add(COleVariant(csFullPath);
  ISCSessionPtr m_scSessionPtr = m_scSessionColPtr->Add();  // add a new session
  CComVariant varResult = m_scSessionPtr->Open(m_scPUnitPtr, SCD_SL_M0); // open persistence
  //unit
  if (varResult.vt == VT_BOOL && varResult.boolVal == FALSE)
    return;
  ISCModelObjectCollectionPtr m_scModelObjColPtr = m_scSessionPtr->GetModelObjects();
  ISCModelObjectPtr rootObj = m_scModelObjColPtr->GetRoot();
  ISCModelPropertyCollectionPtr propColPtr = rootObj->GetProperties();
  ISCModelPropertyPtr pxProp = propColPtr->GetItem("PMFilename");
  VARIANT vtEmpty;
  vtEmpty.vt = VT_ERROR;
  vtEmpty.scode = DISP_E_PARAMNOTFOUND;
  pxProp->PutValue(vtEmpty, (long) SCVT_BSTR, csFullPath);

  // …
}
```

Example: Open a Template Model in Visual Basic

```
Public Sub OpenTemplateModel(persUnitCol as SCAPI.PersistenceUnits, _
  scSessionCol as SCAPI.Sessions, newFilePath as String )

  Dim newModel As SCAPI.PersistenceUnit
  Dim modelObject As SCAPI.ModelObject
  Dim scSession As SCAPI.Session
  Dim fileNameProp as SCAPI.ModelProperty

  Set newModel = persUnitCol.Add(fullModelPath)  ' open the template model
  Set scSession - scSessionCol.Add  ' new session
  bRetVal = scSession.Open(scPersistenceUnit, SCD_SL_M0)  ' connect to a session
  Set modelObject = scSession.ModelObjects.Root  ' this is the PMModel object
  Set fileNameProp = scObj.Properties.Item("PMFilename")  ' get the filename
  property
  fileNameProp.Value = newFilePath  ' set the filename property to its new value
  '…
End Sub
```

Accessing Objects in a Model

You can access model objects through the *ModelObjects* property in an active *ISCSession* instance. The *ModelObjects* property is a collection of all model objects associated with the persistence unit of the session. The *ModelObjects* property is an instance of the *ISCModelObjectCollection*. Iteration through an instance of *ISCModelObjectCollection* is done in a depth-first fashion, and returns instances of *ISCModelObject*.

Interfaces Used to Access Model Objects

The following sections describe the interfaces used to access model objects.

ISCSession Interface

The following table contains information on the *ISCSession* interface:

Signature	Description	Valid Arguments
ISCModelObjectCollection * ModelObjects()	Creates a ModelObject collection for the session	None

ISCModelObjectCollection Interface

The following table contains information on the *ISCModelObjectCollection* interface:

Signature	Description	Valid Arguments
long Count()	Number of objects in the collection	None
IUnknown _NewEnum()	Constructs an instance of the collection enumerator object	None

ISCModelObject Interface

The following table contains information on the *ISCModelObject* interface:

Signature	Description	Valid Arguments
BSTR ClassName()	Returns the class name of the current object	None
SC_OBJID ObjectId()	Uniquely identifies the current object	None

Signature	Description	Valid Arguments
BSTR Name()	Returns the name or a string identifier of the current object	None
SC_CLSID ClassId()	Returns the class identifier of the current object	None
ISCModelObject * Context()	Passes back the context (parent) of the object	None

The following examples illustrate how to access model objects:

Example: Access Model Objects in C++

```
void APIExample::IterateObjects(ISCSessionPtr m_scSessionPtr)
{
   ISCModelObjectCollectionPtr m_scModelObjColPtr = m_scSessionPtr->GetModelObjects();
   IUnknownPtr _NewEnum = NULL;
   IEnumVARIANT* ObjCollection;

   _NewEnum = m_scModelObjColPtr ->Get_NewEnum();
   if (_NewEnum != NULL)
   {
     HRESULT hr = _NewEnum->QueryInterface(IID_IEnumVARIANT, (LPVOID*) &ObjCollection);
     if (!FAILED(hr))
     {
       COleVariant xObject;
       while (S_OK == ObjCollection->Next(1,&xObject,NULL))
       {
          ISCModelObjectPtr pxItem = (V_DISPATCH (&xObject));
          xObject.Clear();
          // Process the Item
          CString csName = (LPSTR) pxItem->GetName();
          CString csID = (LPSTR) pxItem->GetObjectId();
          CString csType = (LPSTR) pxItem->GetClassName();
          // …
       }
     }
     if (ObjCollection)
          ObjCollection->Release();
   }
}
```

Example: Access Model Objects in Visual Basic

```
Public Sub IterateObjects(scSession As SCAPI.Session)
    Dim scModelObjects As SCAPI.ModelObjects
    Dim scObj as SCAPI.ModelObject

    Set scModelObjects - scSession.ModelObjects

    For Each scObj In scModelObjects
        Debug.Print scObj.Name
        Debug.Print scObj.ObjectId
        Dubug.Print scObj.ClassName
    Next

End Sub
```

Accessing a Specific Object

You can directly access model objects in an *ISCModelObjectCollection* instance by using the *Item* method of the interface.

ISCModelObjectCollection Interface

The following table contains information on the *ISCModelObjectCollection* interface:

Signature	Description	Valid Arguments
ISCModelObject * Item(VARIANT nIndex, VARIANT Class [optional])	Returns an IUnknown pointer for a Model Object component identified by the nIndex parameter	Index: ■ VT_UNKNOWN – Pointer to the *ISCModelObject* interface. Given object is returned from the collection. ■ VT_BSTR – ID of an object. The object with the given ID is returned from the collection. ■ VT_BSTR – Name of an object. If the name of an object is used, the *Class* parameter must also be used. The object with the given name and given *Class* type is returned from the collection. Class: ■ Empty – The object specified by *nIndex* is returned from the collection. ■ VT_BSTR – Name of a class. Must be used if the *nIndex* parameter is the name of an object. Returns the object with the given name and given *Class*. For valid object class names, see the Identifiers (see page 172) section. ■ VT_BSTR – Class ID of object type. Must be used if the *nIndex* parameter is the name of an object. Returns the object with the given name and given *Class*. For valid object class IDs, see the Identifiers (see page 172) section.

The following examples illustrate how to access a specific object:

Example: Access a Specific Object in C++

```
void APIExample::GetObject(ISCSessionPtr m_scSessionPtr, CString csID)
{
   ISCModelObjectCollectionPtr m_scModelObjColPtr = m_scSessionPtr->GetModelObjects();
   ISCModelObjectPtr scObj = m_scModelObjColPtr->GetItem(COleVariant(csID));
   // …
}
```

Example: Access a Specific Object in Visual Basic

```
Public Sub GetObject(scSession as SCAPI.Session, objID as String)
   Dim scObjCol as SCAPI.ModelObjects
   Dim scObj as SCAPI.ModelObject

   Set scObjCol = scSession.ModelObjects
   Set scObj = scObjCol.Item(objID)  ' retrieves object with given object ID
End Sub
```

Filtering Object Collections

You can create subsets of an instance of *ISCModelObjectCollection* by using its *Collect* method. The *Collect* method creates a new instance of *ISCModelObjectCollection* based on the filtering criteria specified in the parameters of the method. The filtering criteria is optional, and any number of combinations of criteria can be used.

ISCModelObjectCollection Interface

The following table contains information on the *ISCModelObjectCollection* interface:

Signature	Description	Valid Arguments
ISCModelObjectCollection * Collect(VARIANT Root, VARIANT ClassId [optional], VARIANT Depth [optional], VARIANT MustBeOn [optional], VARIANT MustBeOff [optional])	Creates a *Model Objects* collection, which represents a subcollection of itself	Root: ■ VT_UNKNOWN – *ISCModelObject* pointer of the root object. Returns the descendants of the given object. ■ VT_BSTR – The ID of the root object. Returns the descendants of the object with the given ID. ClassId: ■ VT_ARRAY|VT_BSTR – SAFEARRAY of class IDs. Returns the descendants of the root with the given object class IDs. For valid object IDs, see the Identifiers (see page 172) section.

Signature	Description	Valid Arguments
		■ VT_ARRAY|VT_BSTR – SAFEARRAY of class names. Returns the descendants of the root with the given object class name. For valid object class names, see the Identifiers (see page 172) section. ■ VT_BSTR – Class ID. Returns the descendants of the root with the given object class ID. ■ VT_BSTR – Semicolon delimited list of class IDs. Returns the descendants of the root with the given class IDs. ■ VT_BSTR – Class name. Returns the descendants of the root with the given class name. ■ VT_BSTR – Semicolon delimited list of class names. Returns the descendants of the root with the given class names. ■ Empty – Returns all descendants regardless of class type. Depth: ■ VT_I4 – Maximum depth. Returns the descendants of the root at a depth no more than the given depth. –1 represents unlimited depth. ■ Empty – Returns all descendants of the root (unlimited depth). MustBeOn: ■ VT_I4 – Returns the descendants of the root with the given object flags set. For more information about *SC_ModelObjectFlags*, see the Enumerations (see page 107) section. ■ Empty – Defaults to SCD_MOF_DONT_CARE. MustBeOff: ■ VT_I4 – Returns the descendants of the root that do not have the given object flags set. ■ Empty – Defaults to SCD_MOF_DONT_CARE.

The following sections show the code examples for the different filters.

Example: Object Type Filter in C++

```
void APIExample::FilterObjects(ISCModelObjectPtr rootObj,
        ISCModelObjectCollectionPtr m_scModelObjColPtr, CString csType)
{
   ISCModelObjectCollectionPtr scModelObjects;
   scModelObjects = m_scModelObjColPtr->Collect(rootObj->GetObjectId(), COleVariant(csType));
   // …
}
```

Example: Object Type Filter in Visual Basic

```
Public Sub FilterObjects(rootObj As SCAPI.ModelObject, objCollection As
     SCAPI.ModelObjects, objType as String)

   Dim scModelObjects As SCAPI.ModelObjects
   Set scModelObjects = objCollection.Collect(rootObj, objType as String)
   ' scModelObjects will contain only objects of type objType

End Sub
```

Example: Depth Filter in C++

```
void APIExample::FilterObjects(ISCModelObjectPtr rootObj,
        ISCModelObjectCollectionPtr m_scModelObjColPtr, CString csType, long depth)
{
   ISCModelObjectCollectionPtr scModelObjects;
   scModelObjects = m_scModelObjColPtr->Collect(rootObj->GetObjectId(), COleVariant(csType),
                                                                depth);

   // …
}
```

Example: Depth Filter in Visual Basic

```
Public Sub FilterObjects(rootObj As SCAPI.ModelObject, objCollection As
     SCAPI.ModelObjects, objID As SC_OBJID, depth As Integer)

   Dim scModelObjects As SCAPI.ModelObjects
   Set scModelObjects = objCollection.Collect(rootObj, objID, depth)

End Sub
```

Example: MustBeOn/MustBeOff Filter in C++

```
void APIExample::FilterObjects(ISCModelObjectPtr rootObj,
        ISCModelObjectCollectionPtr m_scModelObjColPtr, CString csType, long depth)
{
   ISCModelObjectCollectionPtr scModelObjects;
   scModelObjects = m_scModelObjColPtr->Collect(rootObj->GetObjectId(), COleVariant(csType),
                                                depth, SCD_MOF_USER_DEFINED);

   // …
}
```

Example: MustBeOn/MustBeOff Filter in Visual Basic

```
Public Sub FilterObjects(rootObj As SCAPI.ModelObject, objCollection As
    SCAPI.ModelObjects, objID As SC_OBJID, depth As Integer)

   Dim scModelObjects As SCAPI.ModelObjects
   Set scModelObjects = objCollection.Collect(rootObj, objID, depth,
                                              SCD_MOF_USER_DEFINED)

End Sub
```

Accessing Object Properties

You can access the properties of an object through the *Properties* property of *ISCModelObject*. The *Properties* property is an instance of *ISCModelPropertyCollection*. The *ISCModelPropertyCollection* contains instances of *ISCModelProperty*.

The following sections describe the interfaces involved with iteration of properties.

ISCModelObject Interface

The following table contains information on the *ISCModelObject* interface:

Signature	Description	Valid Arguments
ISCModelPropertyCollection * Properties()	Returns a property collection of all available properties	None

ISCModelPropertyCollection Interface

The following table contains information on the *ISCModelPropertyCollection* interface:

Signature	Description	Valid Arguments
long Count()	Number of properties in the collection	None
IUnknown _NewEnum()	Constructs an instance of the collection enumerator object	None

ISCModelProperty Interface

The following table contains information on the *ISCModelProperty* interface:

Signature	Description	Valid Arguments
BSTR ClassName()	Returns the class name of the property	None
SC_CLSID ClassId()	Returns the class identifier of the property	None
long Count()	Contains the number of values in the property	None
BSTR FormatAsString()	Formats the property value as a string	None

The following examples illustrate iteration of properties:

Example: Iteration of Properties in C++

```
void APIExample::IterateObjectProperties(ISCModelObjectPtr objPtr)
{
  ISCModelPropertyCollectionPtr propColPtr = objPtr->GetProperties();

  // Iterate over the Collection
  IUnknownPtr _NewEnum = NULL;
  IEnumVARIANT* propCollection;

  _NewEnum = propColPtr->Get_NewEnum();
  if (_NewEnum != NULL)
  {
    HRESULT hr = _NewEnum->QueryInterface(IID_IEnumVARIANT, (LPVOID*) &propCollection);
    if (!FAILED(hr))
    {
      COleVariant xObject;
      while (S_OK == propCollection->Next(1,&xObject,NULL))
      {
        ISCModelPropertyPtr pxProp = (V_DISPATCH (&xObject));
        xObject.Clear();
        if (pxProp.GetInterfacePtr())
        {
          CString csPropName = (LPSTR) pxProp->GetClassName();
          CString csPropVal= (LPSTR) pxProp->FormatAsString();
          // …
        }
      } // property iteration
    }
    if (propCollection)
      propCollection->Release();
  }
}
```

Example: Iteration of Properties in Visual Basic

```
Public Sub IterateObjectProperties(scObj as SCAPI.ModelObject)

  Dim scObjProperties as SCAPI.ModelProperties
  Dim scObjProp As SCAPI.ModelProperty
  Set scObjProperties = scObj.Properties
  For Each scObjProp In scObjProperties
    Debug.Print scObjProp.ClassName
    Debug.Print scObjProp.Name
  Next

End Sub
```

Accessing Scalar Property Values

A scalar property is a property that can be represented as a single value. In certain properties of CA ERwin PM, the values can be expressed as a homogeneous array. In those cases, the property can be expressed as either a scalar property or a non-scalar property. For more information about accessing specific properties, see the appendix Access CA ERwin PM Properties (see page 199).

The value of a scalar property is accessed through the *Value* property of the *ISCModelProperty* interface.

ISCModelProperty Interface

The following table contains information on the *ISCModelProperty* interface:

Signature	Description	Valid Arguments
long Count()	Contains the number of values in the property	None
VARIANT Value(VARIANT ValueId [optional], VARIANT ValueType [optional])	Retrieves the indicated property value in the requested format	ValueId: ■ Empty ■ VT_I2 – 0-based index within a homogeneous array. The value of the member indicated by this index is returned. ValueType: ■ Empty – For more information on default return values, see the appendix Access CA ERwin PM Properties (see page 199). ■ SCVT_DEFAULT – For more information on default return values, see the appendix Access CA ERwin PM Properties (see page 199). ■ SCVT_BSTR – Property value returned as a string.

The following examples illustrate how to access scalar property values:

Example: Access Scalar Property Values in C++

```
void APIExample::IterateScalarProperties(ISCModelPropertyPtr scObjProp)
{
  if (scObjProp->GetCount() <= 1)
  {
    _bstr_t bstrPropVal= scObjProp->FormatAsString();
    // …
  }
}
```

Example: Access Scalar Property Values in Visual Basic

```
Public Sub IterateScalarProperties(scObjProp as SCAPI.ModelProperty)

    If (scObjProp.Count <= 1) Then
      If (IsArray(scObjProp.Value)) Then
                For j = LBound(scObjProp.Value) To UBound(scObjProp.Value)
                    Debug.Print CStr(scObjProp.Value(j))
                Next
      Else
                CStr(scObjProp.Value)
      End If
    End If

End Sub
```

Accessing Non-Scalar Property Values

The properties that contain multiple values (either homogeneous or heterogeneous) are non-scalar properties. For more information about specific properties in CA ERwin PM, see the appendix Access CA ERwin PM Properties (see page 199). To access the individual values of a non-scalar property, the *PropertyValues* member of the *ISCModelProperty* interface is used. The *PropertyValues* member is an instance of *ISCPropertyValueCollection*. Each member of *ISCPropertyValueCollection* is an instance of *ISCPropertyValue*. The *ValueId* member of the *ISCPropertyValue* interface identifies the individual property values in a non-scalar property. *ValueId* can either be a 0-based index or the name of the non-scalar property value member if the property type is a structure.

ISCModelProperty Interface

The following table contains information on the *ISCModelProperty* interface:

Signature	Description	Valid Arguments
ISCPropertyValueCollection * PropertyValues()	Returns the values for the property	None

ISCPropertyValueCollection Interface

The following table contains information on the *ISCPropertyValueCollection* interface:

Signature	Description	Valid Arguments
long Count()	Number of values in the collection	None
IUnknown _NewEnum()	Constructs an instance of the collection enumerator object	None

ISCPropertyValue Interface

The following table contains information on the *ISCPropertyValue* interface:

Signature	Description	Valid Arguments
VARIANT ValueId(VARIANT ValueType [optional])	Uniquely identifies the value in a non-scalar property. For more information about non-scalar property value IDs, see the appendix Access CA ERwin PM Properties (see page 199).	ValueType: ■ SCVT_I2 – If the property is non-scalar, the value of the property index is returned. ■ SCVT_I4 – If the property is non-scalar, the value of the property index is returned. ■ SCVT_BSTR – The name of the non-scalar property member if it is available, or else the index of the member is returned. ■ SCVT_DEFAULT – If the property is non-scalar, the value of the property index is returned. ■ Empty – Defaults to SCVT_DEFAULT.
SC_CLSID PropertyClassId()	Returns the class identifier of the current property	None
BSTR PropertyClassName()	Returns the class name of the current property	None

Signature	Description	Valid Arguments
VARIANT Value(VARIANT ValueType [optional])	Converts the current value to the passed value type	ValueType: ■ SCVT_DEFAULT – For more information on the returned value of a given property, see the appendix Access CA ERwin PM Properties (see page 199). ■ SCVT_BSTR – String representation of the property value. ■ VT_I4 – Type of property. For more information on the returned value of a given property, see the appendix Access CA ERwin PM Properties (see page 199). ■ Empty – Defaults to SCVT_DEFAULT.
SC_ValueTypes ValueType()	Passes back the identifier of the value default type	None
SC_ValueTypes ValueIdType()	Passes back the identifier of the value identifier default type	None
SC_ValueTypes * GetSupportedValueTypes()	Groups a list of supported value types and returns it as a SAFEARRAY	None
SC_ValueTypes * GetSupportedValueIdTypes()	Groups a list of supported value types for the current value identifier and returns it as a SAFEARRAY	None

The following examples illustrate how to access non-scalar property values:

Example: Access Non-Scalar Property Values in C++

```
void APIExample::IterateNonScalarProperties(ISCModelPropertyPtr scObjProp)
{
    if (scObjProp->GetCount() > 1)
    {
      ISCPropertyValueCollectionPtr propVals = scObjProp->GetPropertyValues();
      long numVals = propVals->GetCount();
      for (long i = 0; i < numVals; i++)
      {
          ISCPropertyValuePtr propVal = propVals->GetItem(COleVariant(i));
          VARIANT valType;
          V_VT(&valType) = VT_I4;
          V_I4(&valType) = SCVT_BSTR;
          bstr_t bstrPropVal = propVal->GetValue(valType);

          // To determine if the non-scalar property is a homogeneous array or not:
          SC_ValueTypes scIndexType = propVal->GetValueIdType();
          BOOL bIsHomogeneousArray = ((int) scIndexType == SCVT_I4? TRUE:FALSE);

          // If the property value is not an array, its member name can be obtained by:
          _bstr_t bstrValueId = propVal->GetValueId(valType);
          // …
      }
    }
}
```

Example: Access Non-Scalar Property Values in Visual Basic

```
Public Sub IterateNonScalarProperties(scObjProp as SCAPI.ModelProperty)
Dim scPropValue as SCAPI.PropertyValue

    If (scObjProp.Count > 1) Then
      For Each scPropValue In scObjProp.PropertyValues
            If (IsArray(scPropValue.Value)) Then
              For j = LBound(scPropValue.Value) To UBound(scPropValue.Value)
                 Debug.Print CStr(scPropValue.Value()(j))
              Next
            Else
              If (scPropValue.ValueIdType = SCVT_BSTR) Then
                Debug.Print scPropValue.ValueId(SCVT_BSTR),": ", CStr(scPropValue.Value)
              Else
                Debug.Print CStr(scPropValue.Value)
              End If
      Next
    End If
End Sub
```

Accessing a Specific Property

For non-scalar properties, you can directly access individual values by using the *Item* method of *ISCPropertyValueCollection*.

ISCPropertyValueCollection Interface

The following table contains information on the *ISCPropertyValueCollection* interface:

Signature	Description	Valid Arguments
ISCPropertyValue * Item(VARIANT ValueId)	Returns a single value from the property value collection	ValueId: ■ VT_I2 – Index of the member in a non-scalar property. ■ VT_BSTR – Name of a member in a non-scalar property.

Note: For more information on return values, see the appendix Access CA ERwin PM Properties (see page 199).

The following examples illustrate how to access a specific property:

Example: Access a Specific Property in C++

```
// This function retrieves a specific value with the given index from the property with the
// given name.
ISCPropertyValuePtr APIExample::GetPropValue(ISCModelPropertyPtr scObjProp, CString csName, intindex)
{
    ISCModelPropertyCollectionPtr propColPtr = scObjPtr->GetProperties();
    ISCModelPropertyPtr scObjProp = propColPtr->GetItem(COleVariant(csName));
    ISCPropertyValueCollectionPtr propVals = scObjProp->GetPropertyValues();
    return propVals->GetItem(COleVariant(index));
}

// This function retrieves a specific value with the given member name from the property with the
// given name.  This function can only be used with a non-scalar property that has member names
ISCPropertyValuePtr APIExample::GetPropValue2(ISCModelPropertyPtr scObjProp, CString csName, CString
csMemName)
{
    ISCModelPropertyCollectionPtr propColPtr = scObjPtr->GetProperties();
    ISCModelPropertyPtr scObjProp = propColPtr->GetItem(COleVariant(csName));
    ISCPropertyValueCollectionPtr propVals = scObjProp->GetPropertyValues();
    return propVals->GetItem(COleVariant(csMemName));
}
```

Example: Access a Specific Property in Visual Basic

```
' This function retrieves a specific value with the given index from the property with the
' given name.
Public Function GetPropValue(scObj as SCAPI.ModelObject, propName as String, index as Integer) _
    as SCAPI.PropertyValue

    Dim scProp as SCAPI.ModelProperty
    Set scProp = scObj.Properties.Item(propName)
    Set GetPropValue = scProp.PropertyValues.Item(index)

End Function

' This function retrieves a specific value with the given member name from the property with the
' given name.  This function can only be used with a non-scalar property that has member names
Public Function GetPropValue2(scObj as SCAPI.ModelObject, propName as String, memName as
    String) _
    as SCAPI.PropertyValue

    Dim scProp as SCAPI.ModelProperty
    Set scProp = scObj.Properties.Item(propName)
    Set GetPropValue2 = scProp.PropertyValues.Item(memName)

End Function
```

Filtering Properties

Subsets of an instance of *ISCModelPropertyCollection* can be created by using its *CollectProperties* method of *ISCModelObject*. The *CollectProperties* method creates a new instance of *ISCModelPropertyCollection* based on the filtering criteria specified in the parameters of the method. By filtering the property collection, you can retrieve properties of a certain type, properties with specified flags set, or properties that do not have specified flags set. The filtering criteria is optional, and any number of combinations of criteria can be used.

For more information about identifiers used in property types, see the Identifiers (see page 172) section. For more information about specific property flags, see the Enumerations (see page 107) section.

ISCModelObject Interface

The following table contains information on the *ISCModelObject* interface:

Signature	Description	Valid Arguments
ISCModelPropertyCollection * CollectProperties(VARIANT ClassIds [optional], VARIANT MustBeOn [optional], VARIANT MustBeOff [optional])	Returns a property collection of the type that you require	ClassIds: ■ Empty – All properties of the object are returned. ■ VT_ARRAY|VT_BSTR – SAFEARRAY of property type IDs. Returns the properties with the given property type IDs. ■ VT_ARRAY|VT_BSTR – SAFEARRAY of property names. Returns the properties with the given names. ■ VT_BSTR – ID of a property type. Returns the property with the given property type ID. ■ VT_BSTR – Name of a property. Returns the property with the given name. ■ VT_BSTR – List of property type IDs delimited by semicolons. Returns the properties with the given property type IDs. ■ VT_BSTR – List of property names delimited by semicolons. Returns the properties with the given names.
		MustBeOn: ■ Empty – Defaults to SCD_MPF_DONT_CARE and returns all properties. ■ VT_I4 – SC_ModelObjectFlags flags that must be on. Returns the properties with the specified flags set. MustBeOff: ■ Empty – Defaults to SCD_MPF_NULL and returns all properties. ■ VT_I4 – SC_ModelObjectFlags flags that must be off. Returns the properties that do not have the specified flags set.

The following examples illustrate how to filter properties:

Example: Filter Properties in C++

```
ISCModelObjectPtr scObjPtr;
ISCModelPropertyCollectionPtr propColPtr;

propColPtr = scObjPtr->GetProperties();    // no filtering

VARIANT vtEmpty;
vtEmpty.vt = VT_ERROR;
vtEmpty.scode = DISP_E_PARAMNOTFOUND;

VARIANT valType;
V_VT(&valType) = VT_I4;
V_I4(&valType) = SCD_MPF_SCALAR;

propColPtr = scObjPtr->CollectProperties(vtEmpty, valType, vtEmpty);  // scalar properties only

propColPtr = scObjPtr->CollectProperties(vtEmpty, vtEmpty, valType);  // non-scalar properties only
```

Example: Filter Properties in Visual Basic

```
Dim scObj As SCAPI.ModelObject
Dim scObjProperties As SCAPI.ModelProperties

Set scObjProperties = scObj.Properties   ' no filtering

Set scObjProperties = scObj.CollectProperties(, SCD_MPF_SCALAR)  ' scalar properties only

Set scObjProperties = scObj.CollectProperties(, , SCD_MPF_SCALAR)  ' non-scalar properties only
```

Modifying the Model Using Session Transactions

In order to make modifications to a model, session transactions must be used. Prior to making a modification, *BeginTransaction()* must be called. Once all the modifications are completed, *CommitTransaction()* must be called. Nested transactions and the ability to perform rollbacks are not supported. If *BeginTransaction()* is called while a transaction is active, it is viewed just as a continuation of the previous transaction. A session can only have one active transaction at a time.

Begin Transaction

To indicate that a modification to the model is about to occur, *BeginTransaction()* must be called. Since a session can have at most one active transaction, the transaction ID, which is the return value of this function, is the same as the ID of the session.

ISCSession Interface

The following table contains information on the *ISCSession* interface:

Signature	Description	Valid Arguments
VARIANT BeginTransaction()	Opens a transaction on the session. Returns the identifier of the session as the ID of the transaction.	None

The following examples illustrate modifying the model using the *Begin Transaction*:

Example: Use the Begin Transaction to Modify the Model in C++

```
variant_t m_transactionId;   // transaction ID for the session
// ISCSessionPtr m_scSessionPtr;    // this is the pointer to the current session

VariantInit(&m_transactionId);
m_transactionId = m_scSessionPtr->BeginTransaction();

// …
```

Example: Use the Begin Transaction to Modify the Model in Visual Basic

```
Dim m_scTransactionId As Variant
' m_scSession As SCAPI.Session

m_scTransactionId = m_scSession.BeginTransaction
```

Commit Transaction

CommitTransaction() is used to commit the modifications to the in-memory model.

Note: The *Commit* only applies to the in-memory model while the API is running. To persist the modifications, the model must be explicitly saved using the *ISCPersistenceUnit::Save()* function.

ISCSession Interface

The following table contains information on the *ISCSession* interface:

Signature	Description	Valid Arguments
VARIANT_BOOL CommitTransaction(VARIANT TransactionId)	Commits the specified transaction	

The following examples illustrate modifying the model using the *Commit Transaction*:

Example: Use the Commit Transaction to Modify the Model in C++

```
variant_t m_transactionId;   // transaction ID for the session
// ISCSessionPtr m_scSessionPtr;    // this is the pointer to the current

sessionVariantInit(&m_transactionId);
m_transactionId = m_scSessionPtr->BeginTransaction();

// Make modifications to the model here ….

m_scSessionPtr->CommitTransaction(m_transactionId);
```

Example: Use the Commit Transaction to Modify the Model in Visual Basic

```
Dim m_scTransactionId As Variant
' m_scSession As SCAPI.Session

m_scTransactionId = m_scSession.BeginTransaction

' make modifications here …

m_scSession.CommitTransaction m_scTransactionId
```

Creating Objects

ERwin

The first step in creating a new object is to retrieve the *ISCModelObject* instance of the parent of the new object. From the parent of the new object, retrieve its child objects in an instance of *ISCModelObjectCollection*, and then add the new object to the child objects collection.

For more information on valid child types of an object, valid object class names, and object type IDs, see the appendix CA ERwin PM Metamodel (see page 115).

Interfaces Used to Create a New Object

The following section describes the interface used to create new model objects.

ISCModelObjectCollection Interface

The following table contains information on the *ISCModelObjectCollection* interface:

Signature	Description	Valid Arguments
ISCModelObjectCollection * Collect(VARIANT Root, VARIANT ClassId [optional], VARIANT Depth [optional], VARIANT MustBeOn [optional], VARIANT MustBeOff[optional])	Creates a *Model Objects* collection, which represents a subcollection of itself	Root: ■ VT_UNKNOWN – *ISCModelObject* pointer of the root object. Returns the descendants of the given object. ■ VT_BSTR – The ID of the root object. Returns the descendants of the object with the given ID. ClassId: ■ Empty – Not needed when obtaining the children of an object. Depth: ■ VT_I4 – Set depth to 1 when obtaining the immediate children of an object. MustBeOn: ■ Empty – Not needed when obtaining the children of an object. MustBeOff: ■ Empty – Not needed when obtaining the children of an object.
ISCModelObject * Add(VARIANT Class, VARIANT ObjectId [optional])	Adds an object of type *Class* to the model	Class: ■ VT_BSTR – Name of a class. Creates an object of the given class name. ■ VT_BSTR – Class ID of an object type. Creates an object of the type with the given ID. ObjectId: ■ Empty – The API assigns an ID for a new object. ■ VT_BSTR – ID for a new object. The API assigns the given ID to the new object.

The following examples illustrate how to create objects:

Example: Create Objects in C++

```
// NOTE: ISCSession::BeginTransaction() must be called prior to calling this function
//    ISCSession::CommitTransaction() must be called upon returning from this function
void APIExample::CreateObject(CString csType, ISCModelObjectPtr parentObj)
{
  VARIANT vtEmpty;
  vtEmpty.vt = VT_ERROR;
  vtEmpty.scode = DISP_E_PARAMNOTFOUND;

  ISCModelObjectCollectionPtr childObjs = m_scModelObjColPtr->Collect
        (parentObj->GetObjectId(),vtEmpty,(long)1);  // get child objects

  // Add child object to collection
  ISCModelObjectPtr childObj = childObjs->Add(COleVariant(csType));
  // …
}
```

Example: Create Objects in Visual Basic

```
Public Sub AddNewObject(scSession as SCAPI.Session, parentObj as SCAPI.ModelObject, type as String)
    Dim scObj as SCAPI.ModelObject
    Dim m_scChildObjs As SCAPI.ModelObjects
    Dim transactionID as Variant

    transactionID = scSession.BeginTransaction
    m_scChildObjs = scSession.ModelObjects.Collect(parentObj, , 1)  ' child objects collection
    Set scObj = m_scChildObjs.Add(type)   ' add new object to the child object collection

    scSession.CommitTransaction transactionID
End Sub
```

Setting Property Values

To set a property value of a model object, use the *Value* member of an instance of the *ISCModelProperty* interface.

Setting Scalar Property Values

The valid *VARIANT* types that can be used to set a scalar property value is dependent on the type of the property. For more information, see the appendix Access CA ERwin PM Properties (see page 199).

ISCModelProperty Interface

The following table contains information on the *ISCModelProperty* interface:

Signature	Description	Valid Arguments
void Value(VARIANT ValueId [optional], VARIANT ValueType [optional], VARIANT Val)	Sets the indicated property value with the given value	ValueId: ■ Empty – Not used when setting scalar properties. ValueType: ■ Empty – Not used. Val: ■ Dependent upon the property type. For more information on valid values, see the appendix Access CA ERwin PM Properties (see page 199).

The following examples illustrate how to set scalar property values:

Example: Set Scalar Property Values in C++

```
// NOTE: ISCSession::BeginTransaction() must be called prior to calling this function
//    ISCSession::CommitTransaction() must be called upon returning from this function
void APIExample::SetNameProperty(ISCModelObjectPtr objPtr, CString csName)
{
   ISCModelPropertyCollectionPtr propCol = objPtr->GetProperties();
   CString csPropName = "Name";
   ISCModelPropertyPtr nameProp = propCol->GetItem(COleVariant(csPropName));
   if (nameProp != NULL)
      nameProp->PutValue(vtEmpty, (long) SCVT_BSTR, csName);
}
```

Example: Set Scalar Property Values in Visual Basic

```
' NOTE: ISCSession::BeginTransaction() must be called prior to calling this function
'    ISCSession::CommitTransaction() must be called upon returning from this function
Public Sub SetScalarPropValue(modelProp as SCAPI.ModelProperty, val as Variant)
     modelProp.Value = val
End Sub
```

Setting Non-Scalar Property Values

To set a non-scalar property value, you must identify the specific value that you want to set. This is done through the *ValueId* parameter. The *ValueId* can either be the 0-based index of the property value collection or the name of the member if the property is a structure.

For more information on *ValueId*s and the valid *VARIANT* types that can be used with the property values, see the section Non-scalar Access (see page 201).

ISCModelProperty Interface

The following table contains information on the *ISCModelProperty* interface:

Signature	Description	Valid Arguments
void Value(VARIANT ValueId [optional], VARIANT ValueType [optional], VARIANT Val)	Sets the indicated property value with the given value	ValueId: ■ VT_I4 – Index for a non-scalar property of which the given value is set. ■ VT_BSTR – Name of a member in a non-scalar property of which the given value is set. ValueType: ■ Empty – Not used. Val: ■ Dependent upon the property type. For more information on valid values, see the appendix Access CA ERwin PM Properties (see page 199).

The following examples illustrate how to set non-scalar property values:

Example: Set Non-Scalar Property Values in C++

```
// ISCModelPropertyPtr dateProp;

CString csMemberName = "year";
dateProp->PutValue(COleVariant(csMemberName),(long) SCVT_I4, (long) 2000);  // name of member used
csMemberName = "month";
dateProp->PutValue(COleVariant(csMemberName),(long) SCVT_I4, (long) 12);   // name of member used
dateProp->PutValue((long) 2,(long) SCVT_I4, (long) 25);  // index 2 is day
```

Example: Set Non-Scalar Property Values in Visual Basic

```
modelProp.Value(index) = val   ' index is used to access non-scalar property

modelProp.Value(memberName) = val   ' name of member is used to access non-scalar property
```

Deleting Objects

You can delete an object by removing the *ISCModelObject* interface instance of the object from the instance of *ISCModelObjectCollection*. You identify the object that you want to delete either by its pointer to the interface or by its object ID.

Interfaces Used to Delete Objects

The following section describes the interface used to delete model objects.

ISCModelObjectCollection Interface

The following table contains information on the *ISCModelObjectCollection* interface:

Signature	Description	Valid Arguments
VARIANT_BOOL Remove(VARIANT Object)	Removes the specified model object from a model	Object: ■ VT_UNKNOWN – *ISCModelObject* * pointer to the object that you want to delete. Removes the given object. ■ VT_BSTR – ID of the object. Removes the object with the given ID.

The following examples illustrate how to delete objects:

Example: Delete Objects in C++

```
CString csID;   // ID of object to be removed
// …
CComVariant bRetVal = scObjCol->Remove(COleVariant(csID));
```

Example: Delete Objects in Visual Basic

```
bRetVal = scObjCol.Remove(objID)
```

Deleting Properties and Property Values

Properties are deleted by removing the property from the instance of the *ISCModelPropertyCollection* interface. If the property is non-scalar, the individual property value can be removed by using the *RemoveValue* method of the *ISCModelProperty* interface. For more information on valid property names and property IDs, see the Identifiers (see page 172) section.

Interfaces Used to Delete Properties and Property Values

The following sections describe the interfaces used to delete model properties and model property values.

ISCModelPropertyCollection Interface

The following table contains information on the *ISCModelPropertyCollection* interface:

Signature	Description	Valid Arguments
VARIANT_BOOL Remove(VARIANT ClassId)	Removes the indicated property from the bound object	ClassId: ■ VT_UNKNOWN – *ISCModelProperty* pointer to the object that you want to remove. Removes the given property. ■ VT_BSTR – Name of the property. Removes the property with the given name. ■ VT_BSTR – ID of the property. Removes the property with the given ID.

ISCModelProperty Interface

The following table contains information on the *ISCModelProperty* interface:

Signature	Description	Valid Arguments
VARIANT_BOOL RemoveValue(VARIANT ValueId [optional])	Removes the specified value from the property	ValueId: ■ Empty – For scalar properties only. ■ VT_I4 – Index of a non-scalar property. Removes the value with the given index in a non-scalar property. ■ VT_BSTR – Name of the property member in a non-scalar property. Removes the value of the non-scalar property member with the given name.
VARIANT_BOOL RemoveAllValues()	Remove all values from the property	None

Example: Delete Scalar Properties in C++

```
CString propName;
// …
CComVariant bRetVal = m_scObj->GetProperties()->Remove(COleVariant(propName));
```

Example: Delete Scalar Properties in Visual Basic

```
bRetVal = m_scObj.Properties.Remove(propName)
```

Example: Delete Non-Scalar Property Values in C++

```
CString propName;   // name of non-scalar property
// …
CComVariant bRetVal = m_scObj->GetProperties()->Remove(COleVariant(propName));  // removes all vals

long index;   // index of a member in a non-scalar property
// …
bRetVal = m_scModelProperty->RemoveValue(index);   // remove single value from the property
```

Example: Delete Non-Scalar Property Values in Visual Basic

```
bRetVal = m_scObj.Properties.Remove(propName) ' Remove all values from the property

bRetVal = modelProp.RemoveValue(index)  ' Remove single value from the property
```

Saving the Model

If modifications were made to the CA ERwin PM model, the persistence unit must be saved in order to persist the changes.

ISCPersistenceUnit Interface

The following table contains information on the *ISCPersistenceUnit* interface:

Signature	Description	Valid Arguments
VARIANT_BOOL Save(VARIANT Locator [optional], VARIANT Disposition [optional]	Persists model data to external storage	Locator: ■ VT_BSTR – Full path of the location to store the model. ■ Empty – The location specified by the *PMFilename* property of the *PMModel* object is used to store the model. ■ Disposition – Not used.

The following examples illustrate how to save a model:

Example: Save the Model in C++

```
// ISCPersistenceUnitPtr m_scPUnitPtr;
ISCPropertyBagPtr propBagPtr = m_scPUnitPtr->GetPropertyBag ("Locator");
long index = 0;
_bstr_t bstrFileName = propBagPtr->GetValue(COleVariant(index));
m_scPUnitPtr->Save(bstrFileName);
```

Example: Save the Model in Visual Basic

```
' Dim scPUnit as SCAPI.PersistenceUnit

Dim propBag As SCAPI.PropertyBag
Set propBag = scPUnit.PropertyBag("Locator")  ' gets the filename of the persistence unit
scPUnit.Save propBag.Value(0)
```

Accessing Metamodel Information

You can obtain metamodel information using the API. The metamodel can be accessed in the same manner as a model with the following exceptions:

- A persistence unit that is associated with the session is NULL.
- The metamodel information cannot be modified.

To indicate that a session will access metamodel information, the *Level* parameter of the *Open* method is set to SCD_SL_M1.

In the metamodel, there are three classes of objects:

Model

Specifies the root of the metamodel objects. Model represents the CA ERwin PM model.

Model Object

Specifies the different types of objects within CA ERwin PM.

Model Property

Specifies the properties of the model objects.

When you use the *ISCModelObjectCollection::Collect* function in the metamodel mode, the three classes listed above are the valid choices for the *ClassId* parameter. When you obtain properties using the *ISCModelObject::Properties* or *ISCModelObject::CollectProperties* functions, meta-properties of the given object type are returned. To obtain the properties associated with a given object type, you need to use the *ISCModelObjectCollection::Collect* function with the object as the root, and *Model Property* as the *ClassId* parameter.

ISCSession Interface

The following table contains information on the *ISCSession* interface:

Signature	Description	Valid Arguments
VARIANT_BOOL Open(ISCPersistenceUnit * Unit, VARIANT Level [optional], VARIANT Flags [optional])	Binds self to the persistence unit identified by the Unit parameter	Unit: ■ Pointer to a NULL persistence unit Level: ■ SCD_SL_M1 – Metamodel access Flags: ■ Empty – Defaults to SCD_SF_NONE

The following examples illustrate how to access metamodel information:

Example: Access Metamodel Information in C++

```
void APITest::IterateMetaModel()
{
  ISCPersistenceUnitPtr tempUnit = NULL;

  m_scSessionColPtr = m_scAppPtr->GetSessions();
  m_scSessionPtr = m_scSessionColPtr->Add();  // add a new session
  CComVariant varResult = m_scSessionPtr->Open(tempUnit, (long) SCD_SL_M1);  // meta-model level
  if (varResult.vt == VT_BOOL && varResult.boolVal == FALSE)
    return;
  m_scMetaObjColPtr = m_scSessionPtr->GetModelObjects();
  PrintObjInfo(m_scMetaObjColPtr->GetRoot());
  PrintChildObjs(m_scMetaObjColPtr->GetRoot());
}
```

```
void APITest::PrintObjInfo(ISCModelObjectPtr scObj)
{
  CString csName = (LPSTR) scObj->GetName();
  CString csID = scObj->GetObjectId();
  CString csType = (LPSTR) scObj->GetClassName();
  TRACE("%s: %s (%s) □n", csType, csName, csID);
  ISCModelPropertyCollectionPtr propColPtr = scObj->GetProperties();  // meta-properties
  IUnknownPtr _NewEnum = NULL;
  IEnumVARIANT* propCollection;
  _NewEnum = propColPtr->Get_NewEnum();
  if (_NewEnum != NULL)
  {
    TRACE("Meta-properties:□n");
    HRESULT hr = _NewEnum->QueryInterface(IID_IEnumVARIANT, (LPVOID*) &propCollection);
    if (!FAILED(hr))
    {
      COleVariant xObject;
      while (S_OK == propCollection->Next(1,&xObject,NULL))
      {
            ISCModelPropertyPtr pxProp = (V_DISPATCH (&xObject));
            xObject.Clear();
            if (pxProp.GetInterfacePtr())
            {
              CString csPropName = (LPSTR) pxProp->GetClassName();
              CString csPropVal= (LPSTR) pxProp->FormatAsString();
              TRACE("    %s: %s□n", csPropName, csPropVal);
            } // meta-property iteration
      }
      if (propCollection)
        propCollection->Release();
    }
  }
  CString csObjType = "Model Property";     // Print the properties of the object
  ISCModelObjectCollectionPtr propCol = m_scMetaObjColPtr->Collect(scObj->GetObjectId(),
  COleVariant(csObjType, (long) 1);
```

```
  NewEnum = propCol->Get_NewEnum();
  if (_NewEnum != NULL)
  {
    TRACE("Properties: \n");
    HRESULT hr = _NewEnum->QueryInterface(IID_IEnumVARIANT, (LPVOID*)
    &propCollection);
    if (!FAILED(hr))
    {
      COleVariant xObject;
      while (S_OK == propCollection->Next(1,&xObject,NULL))
      {
            ISCModelObjectPtr pxProp = (V_DISPATCH (&xObject));
            xObject.Clear();
            if (pxProp.GetInterfacePtr())
            {
              CString csPropName = (LPSTR) pxProp->GetName();
              TRACE("    %s\n", csPropName);
            } // property iteration
      }
      if (propCollection)
        propCollection->Release();
    }
  }
}
void APITest::PrintChildObjs(ISCModelObjectPtr rootObj)
{
  IUnknownPtr _NewEnum = NULL;
  IEnumVARIANT* ObjCollection;
  CString csObjType = "Model Object";

  ISCModelObjectCollectionPtr childObjs = m_scMetaObjColPtr->Collect(rootObj->GetObjectId(),
  COleVariant(csObjType));

  _NewEnum = childObjs->Get_NewEnum();
  if (_NewEnum != NULL)
  {
    HRESULT hr = _NewEnum->QueryInterface(IID_IEnumVARIANT,
    (LPVOID*) &ObjCollection);
    if (!FAILED(hr))
    {
      COleVariant xObject;
      while (S_OK == ObjCollection->Next(1,&xObject,NULL))
      {
            ISCModelObjectPtr pxItem = (V_DISPATCH (&xObject));
            xObject.Clear();
            PrintObjInfo(pxItem);
      }
```

```
      if (ObjCollection)
        ObjCollection->Release();
    }
  }
}
```

Example: Access Metamodel Information in Visual Basic

```
Dim m_scMetaModelSession As SCAPI.Session
Dim m_scModelObjCol As SCAPI.ModelObjects

Public Sub AccessMetaModel(m_scSessionCol As SCAPI.Sessions)

    Dim scTempPU As SCAPI.PersistenceUnit ' null persistence unit
    Dim objRoot As SCAPI.ModelObject

    Set m_scMetaModelSession = m_scSessionCol.Add  ' new session
    bRetVal = m_scMetaModelSession.Open(scTempPU, SCD_SL_M1)

    ' Model objects
    Set m_scModelObjCol = m_scMetaModelSession.ModelObjects
    Set objRoot = m_scModelObjCol.Root  ' root of collection is the PM model
    PrintObjInfo objRoot
    PrintChildObjs objRoot
End Sub

Sub PrintObjInfo(obj AS SCAPI.ModelObject)
  Dim ObjProperties As SCAPI.ModelObjects
  Dim childObjs As SCAPI.ModelObjects

  Debug.Print "Model Object: "; obj.Name
  Debug.Print "  Class name: "; obj.ClassName; "  (ID: "; obj.ClassId; ")"
  Debug.Print "  Id: "; obj.ObjectId

  ' Output the properties of the object
  Set ObjProperties = m_scModelObjCol.Collect(obj, "Model Property", 1)
  Debug.Print "  Properties: ("; ObjProperties.Count; ")"
  For Each ObjProperty In ObjProperties
      Debug.Print ObjProperty.Name; " - "; ObjProperty.ObjectId
  Next
End Sub
```

```
Sub PrintChildObjs(rootObj As SCAPI.ModelObject)
    Dim objCol As SCAPI.ModelObjects
    Dim obj As SCAPI.ModelObject
    Dim chilObj As SCAPI.ModelObject

    Set objCol = m_scModelObjCol.Collect(rootObj, "Model Object")
    For Each obj In objCol    ' iterate through child objects or the root
        PrintObjInfo obj

        ' Output the properties of the object
        Set ObjProperties = m_scModelObjCol.Collect(obj, "Model Property", 1)
        Debug.Print "  Properties: ("; ObjProperties.Count; ")"
        For Each ObjProperty In ObjProperties
            Debug.Print ObjProperty.Name; " - "; ObjProperty.ObjectId
        Next

        ' Output the child objects
        Set childObjs = m_scModelObjCol.Collect(obj, "Model Object", 1)
        For Each childObj In childObjs
            Debug.Print "    "; childObj.Name; " - "; childObj.ObjectId
        Next
    Next
End Sub
```

Closing the API

When the client of the API has finished accessing the model, the sessions that were open must be closed, and the persistence unit collection must be cleared.

ISCSession Interface

The following table contains information on the *ISCSession* interface:

Signature	Description	Valid Arguments
VARIANT_BOOL Close()	Disconnects self from its associated persistence unit	None

ISCSessionCollection Interface

The following table contains information on the *ISCSessionCollection* interface:

Signature	Description	Valid Arguments
VARIANT_BOOL Remove(VARIANT SessionId)	Removes a Session object from the collection	SessionId: ■ VT_UNKNOWN – Pointer to the *ISCSession* interface. Removes the given session from the collection. ■ VT_I4 – Zero-based index in the session collection. Removes the session with the given index from the collection.

The following examples illustrate how to close a session:

Example: Close the Session in C++

```
// close the sessions
m_scSessionPtr->Close();    // close a single session
m_scSessionColPtr->Clear();   // clear the collection of sessions
```

Example: Close the Session in Visual Basic

```
For Each scSession In m_scSessionCol
    scSession.Close
Next
While (m_scSessionCol.Count > 0)
     m_scSessionCol.Remove (0)
End
```

ISCPersistenceUnitCollection Interface

The following table contains information on the *ISCPersistenceUnitCollection* interface:

Signature	Description	Valid Arguments
VARIANT_BOOL Clear()	Purges all units from the collection	None

The following examples illustrate how to clear persistence units:

Example: Clear Persistence Units in C++

```
// remove the persistence units
m_scPUnitColPtr->Clear();
```

Example: Clear Persistence Units in Visual Basic

```
m_scAppPtr.PersistenceUnits.Clear
```

Error Handling

The CA ERwin PM API uses exception handling to handle errors. To ensure a stable application, it is recommended that API clients use exception handling to trap potential errors such as attempting to access an object that was deleted, or attempting to access an empty collection.

The following examples illustrate error handling:

Example: C++ Error Handling

```
long APIExample::GetObjectProperties(ISCModelObjectPtr objPtr)
{
        // Get the collection of Properties
        ISCModelPropertyCollectionPtr pxPropCol;
        try
        {
                pxPropCol = objPtr->GetProperties();
                if (!pxPropCol.GetInterfacePtr())
                {
                  AfxMessageBox("Unable to Get Properties Collection");
                  return FALSE;
                }
                // …
        }
        catch(_com_error &error)
        {
             AfxMessageBox(error.Description());
        }
}
```

Example: Visual Basic Error Handling

```
Public Sub GetObject(scSession as SCAPI.Session, objID as String)
   Dim scObjCol as SCAPI.ModelObjects
   Dim scObj as SCAPI.ModelObject

   On Error GoTo ErrExit    ' Exception Handling

   Set scObjCol = scSession.ModelObjects
   Set scObj = scObjCol.Item(objID)  ' retrieves object with given object ID

ErrExit:
    MsgBox ("PM_SCAPI Failure.  " + Err.Description)

End Sub
```

Appendix A: API Interfaces Reference

This appendix lists the interfaces contained in the CA ERwin PM API, together with the methods and arguments associated with these interfaces. There is also a section that contains information regarding enumerations.

This section contains the following topics:

API Interfaces

This section describes each API interface, and the methods associated with them. Where applicable, signatures and valid arguments are also described.

Note: Some parameters contain an [optional] designation-this means that this particular part of the parameter is optional and not required.

ISCApplication

The *ISCApplication* interface is the entry point for the API client. *ISCApplication* holds a list of available models (persistence units), and connections (sessions) between the API client and the models.

The following table contains the methods for the *ISCApplication* interface:

Method	Description
BSTR Name()	Modeling tool application name
BSTR Version()	Modeling tool application version
BSTR ApiVersion()	The API version
ISCApplicationEnvironment * ApplicationEnvironment()	Reports attributes of runtime environment and available features, such as add-in mode, user interface visibility, and so on
ISCApplicationServiceCollection * ApplicationServices()	Not used in the CA ERwin PM API
ISCApplicationWindowCollection * ApplicationWindows()	Not used in the CA ERwin PM API
ISCModelDirectoryCollection * ModelDirectories()	Not used in the CA ERwin PM API

Method	Description
ISCPersistenceUnitCollection * PersistenceUnits()	Returns a collection of all persistence units loaded in the application
ISCSessionCollection * Sessions()	Returns a collection of sessions created within the application

ISCApplicationEnvironment

The *ISCApplicationEnvironment* interface contains the information about the runtime environment.

The following table contains the methods for the *ISCApplicationEnvironment* interface:

Method	Description
ISCPropertyBag * PropertyBag(VARIANT Category [optional], VARIANT Name [optional], VARIANT AsString [optional]	Populates a property bag with one or more property values as indicated by *Category* and *Name*.

ISCApplicationEnvironment::PropertyBag Arguments

Here is the signature for the *PropertyBag* function:

```
ISCPropertyBag *PropertyBag(VARIANT Category, VARIANT Name, VARIANT AsString)
```

The following table contains the valid arguments for the *PropertyBag* function:

Parameter	Valid Type/Value	Description
Category [optional]	Empty	Complete set of features from all categories are returned
	VT_BSTR – name of category	Features from the given category are returned
Name [optional]	Empty	All properties from the selected category are returned
	VT_BSTR	The property with the given name and category is returned

Parameter	Valid Type/Value	Description
AsString [optional]	Empty	All values in the property bag are presented in native type
	VT_BOOL	If set to TRUE, all values in the property bag are presented as strings

ISCModelObject

The *ISCModelObject* interface represents an object in a model.

The following table contains the methods for the *ISCModelObject* interface:

Method	Description
BSTR ClassName()	Returns the class name of the current object.
SC_OBJID ObjectId()	Uniquely identifies the current object.
BSTR Name()	Returns the name or a string identifier of the current object.
SC_CLSID ClassId()	Returns the class identifier of the current object.
ISCModelObject * Context()	Passes back the context (parent) of the object.
SC_ModelObjectFlags Flags()	Returns the flags of the object. For information on SC_ModelObjectFlags, see the Enumerations (see page 107) section.
ISCModelPropertyCollection * Properties()	Returns a property collection of all available properties.
ISCModelPropertyCollection * CollectProperties(VARIANT ClassIds [optional], VARIANT MustBeOn [optional], VARIANT MustBeOff [optional])	Returns a property collection of the type that you want. This method always returns a valid collection even if the collection is empty.
VARIANT_BOOL IsInstanceOf(VARIANT ClassId)	Returns TRUE if self is an instance of the passed class. This method respects inheritance. If *ClassId* contains an ancestor class, the method returns TRUE.
VARIANT_BOOL IsValid()	Returns TRUE if self is valid. This method is used to detect if the referenced object is deleted.

ISCModelObject::CollectProperties Arguments

Here is the signature for the *CollectProperties* function:

```
ISCModelPropertyCollection * CollectProperties(VARIANT ClassIds, VARIANT MustBeOn, VARIANT MustBeOff)
```

The following table contains the valid arguments for the *CollectProperties* function:

Parameter	Valid Type/Value	Description
ClassIds [optional]	Empty	All properties of the object are returned
	VT_ARRAY|VT_BSTR – SAFEARRAY of property IDs	Returns the properties with the given IDs.
	VT_ARRAY|VT_BSTR – SAFEARRAY of property names	Returns the properties with the given names.
	VT_BSTR – ID of a property	Returns the property with the given ID.
	VT_BSTR – name of a property	Returns the property with the given name.
MustBeOn [optional]	Empty	Defaults to SCD_MPF_DONT_CARE – returns all properties
	VT_I4 – SC_ModelObjectFlags flags that must be on	Returns the properties with the specified flags set.
MustBeOff [optional]	Empty	Defaults to SCD_MPF_NULL – returns all properties
	VT_I4 – SC_ModelObjectFlags flags that must be off	Returns the properties that do not have the specified flags.

Note: For more information about valid IDs and valid property names, see the Identifiers (see page 172) section. For more information on SC_ModelObjectFlags, see the Enumerations (see page 107) section.

ISCModelObject::IsInstanceOf Arguments

Here is the signature for the *IsInstanceOf* function:

```
VARIANT_BOOL IsInstanceOf(VARIANT ClassId)
```

The following table contains the valid arguments for the *IsInstanceOf* function:

Parameter	Valid Type/Value	Description
ClassId	VT_BSTR – ID of an object type	Returns TRUE if the object is the same type as the given object type.
	VT_BSTR – name of an object type	Returns TRUE if the object is the same type as the given object type.

Note: For more information about valid IDs and valid object names, see the Identifiers (see page 172) section.

ISCModelObjectCollection

The *ISCModelObjectCollection* interface is a collection of objects in the model that is connected to the active session. Membership in this collection can be limited by establishing filter criteria.

The following table contains the methods for the *ISCModelObjectCollection* interface:

Method	Description
ISCModelObject * Item(VARIANT nIndex, VARIANT Class [optional])	Returns an *IUnknown* pointer for a *Model Object* component identified by the Index parameter.
long Count()	Number of objects in the collection. The number does not include the root object.
IUnknown _NewEnum()	Constructs an instance of the collection enumerator object.
ISCModelObject * Root()	Returns a pointer to the root object in a collection.
SC_CLSID * ClassIds()	Returns a SAFEARRAY of class identifiers (such as object type IDs) within the collection.
BSTR * ClassNames()	Returns a SAFEARRAY of class names (such as object type names) within the collection.

Method	Description
long Depth()	Depth limit on iteration in the collection. -1 represents unlimited depth.
SC_ModelObjectFlags MustBeOn()	Filter on model object flags in the collection
SC_ModelObjectFlags MustBeOff()	Filter on model object flags in the collection
ISCModelObjectCollection * Collect(VARIANT Root, VARIANT ClassId [optional], VARIANT Depth [optional], VARIANT MustBeOn [optional], VARIANT MustBeOff [optional])	Creates a *Model Objects* collection, which represents a subcollection of itself. The method creates a valid collection even though the collection may be empty.
ISCModelObject * Add(VARIANT Class, VARIANT ObjectId)	Adds an object of type *Class* to the model.
VARIANT_BOOL Remove(VARIANT Object)	Removes the specified model object from a model.

ISCModelObjectCollection::Collect Arguments

Here is the signature for the *Collect* function:

ISCModelObjectCollection * Collect(VARIANT Root, VARIANT ClassId, VARIANT Depth, VARIANT MustBeOn, VARIANT MustBeOff)

The following table contains the valid arguments for the *Collect* function:

Parameter	Valid Type/Value	Description
Root	VT_UNKNOWN – ISCModelObject pointer of the root object	Returns the descendants of the given object
	VT_BSTR – the ID of the root object	Returns the descendants of the object with the given ID
ClassId [optional]	VT_ARRAY|VT_BSTR – SAFEARRAY of class IDs	Returns the descendants of the root with the given object class IDs.
	VT_ARRAY|VT_BSTR – SAFEARRAY of class names	Returns the descendants of the root with the given object class name.
	VT_BSTR – class ID	Returns the descendants of the root with the given object class ID.
	VT_BSTR – semicolon delimited list of class IDs	Returns the descendants of the root with the given class IDs.

Parameter	Valid Type/Value	Description
	VT_BSTR – class name	Returns the descendants of the root with the given class name.
	VT_BSTR – semicolon delimited list of class names	Returns the descendants of the root with the given class names.
	Empty	Returns all descendants regardless of class type.
Depth [optional]	VT_I4 – maximum depth for descendants. Depth of 1 returns the immediate children of the root. A depth of -1 (which is the default value) represents unlimited depth.	Returns the descendants of the root at a depth no more than the given depth.
	Empty	Returns all descendants of the root (unlimited depth).
MustBeOn [optional]	VT_I4 – SC_ModelObjectFlags that must be set	Returns the descendants of the root with the given object flags set.
	Empty	Defaults to SCD_MOF_DONT_CARE.
MustBeOff [optional]	VT_I4 – SC_ModelObjectFlags that must not be set	Returns the descendants of the root that do not have the given object flags set.
	Empty	Defaults to SCD_MOF_DONT_CARE.

Note: For more information about valid object IDs and valid object class names, see the Identifiers (see page 172) section. For more information about SC_ModelObjectFlags, see the Enumerations (see page 107) section.

ISCModelObjectCollection::Item Arguments

Here is the signature for the *Item* function:

```
ISCModelObject * Item(VARIANT nIndex,  VARIANT Class)
```

The following table contains the valid arguments for the *Item* function:

Parameter	Valid Type/Value	Description
nIndex	VT_UNKNOWN – pointer to ISCModelObject interface	Returns the given object.
	VT_BSTR – ID of an object	Returns the object with the given ID.
	VT_BSTR – name of an object	If the name of an object is used, the *Class* parameter must also be used. Returns the object with the given name and given *Class* type.
Class [optional]	Empty	Returns the object specified by *nIndex.*
	VT_BSTR – name of a class	Must be used if the *nIndex* parameter is the name of an object. Returns the object with the given name and given *Class*.
	VT_BSTR – class ID of object type	Must be used if the *nIndex* parameter is the name of an object. Returns the object with the given name and given *Class*.

Note: For more information about valid object class names, see the Identifiers (see page 172) section.

ISCModelObjectCollection::Add Arguments

Here is the signature for the *Add* function:

```
ISCModelObject * Add(VARIANT Class, VARIANT ObjectId)
```

The following table contains the valid arguments for the *Add* function:

Parameter	Valid Type/Value	Description
Class	VT_BSTR – name of a class	Creates an object of the given class name.
	VT_BSTR – class ID of an object type	Creates an object of the type with the given ID.
ObjectId [optional]	Empty	API assigns ID for a new object
	VT_BSTR – object ID for a new object	API assigns the given ID to the new object

Note: For more information about valid object class names and valid object class IDs, see the Identifiers (see page 172) section.

ISCModelObjectCollection::Remove Arguments

Here is the signature for the *Remove* function:

```
VARIANT_BOOL Remove(VARIANT Object)
```

The following table contains the valid arguments for the *Remove* function:

Parameter	Valid Type/Value	Description
Object	VT_UNKNOWN – ISCModelObject - pointer to an object	Removes the given object
	VT_BSTR – ID of the object	Removes the object with the given ID

ISCModelProperty

The *ISCModelProperty* interface represents a property of a given object.

The following table contains the methods for the *ISCModelProperty* interface:

Method	Description
BSTR ClassName()	Returns the class name of the property.
SC_CLSID ClassId()	Returns the class identifier of the property.
long Count()	Contains the number of values in the property.
SC_ModelPropertyFlags Flags()	Returns the flags of the property. For information about SC_ModelPropertyFlags, see the Enumerations (see page 107) section.
ISCPropertyValueCollection * PropertyValues()	Returns the collection of values for the model property.
VARIANT Value(VARIANT ValueId [optional], VARIANT ValueType [optional])	Retrieves the indicated property value in the requested format.
void Value(VARIANT ValueId [optional], VARIANT ValueType [optional], VARIANT Val)	Sets the indicated property value with the given value.
SC_ValueTypes DataType(VARIANT ValueId [optional])	Passes back the identifier of the default value type for the indicated property value.
BSTR FormatAsString()	Formats the property value as a string.
VARIANT_BOOL RemoveValue(VARIANT ValueId [optional])	Removes the specified value from the property.
VARIANT_BOOL RemoveAllValues()	Remove all values from the property.
VARIANT_BOOL IsValid()	Returns TRUE if self is valid.

ISCModelProperty::Value Arguments (Get Function)

Here is the signature for the *Value (Get)* function:
VARIANT Value(VARIANT ValueId, VARIANT ValueType)

The following table contains the valid arguments for the *Value (Get)* function:

Parameter	Valid Type/Value	Description
ValueId [optional]	Empty	Ignored if the property is scalar. The return value depends on the property type.
	VT_BSTR – name of a non-scalar member or property	Ignored if the property is scalar. If the property is non-scalar, the value of the member indicated by this name is returned.
	VT_I2 – index of a non-scalar member or property	Ignored if the property is scalar. If the property is non-scalar, the value of the member indicated by this index is returned.
ValueType [optional]	Empty	See note below.
	VT_I4 – SCVT_DEFAULT	See note below.
	VT_I4 – SCVT_BSTR	Property value returned as a string.
	VT_I4 – SCVT_DEFAULT	See note below.

Note: For more information on valid non-scalar cases, default return values, and mapped return values, see the appendix Access CA ERwin PM Properties (see page 199).

ISCModelProperty::Value Arguments (Set Function)

Here is the signature for the *Value (Set)* function:
void Value(VARIANT ValueId, VARIANT ValueType, VARIANT Val)

The following table contains the valid arguments for the *Value (Set)* function:

Parameter	Valid Type/Value	Description
ValueId [optional]	Empty	For scalar properties only
	VT_I4 – index of a non-scalar property	Sets the value with the given index in a non-scalar property

Parameter	Valid Type/Value	Description
	VT_BSTR – name of the property member in a non-scalar property	Sets the value of the non-scalar property member with the given name
ValueType [optional]	Empty	Not used
Val	Dependent upon the property type	For valid values, see the appendix Access CA ERwin PM Properties (see page 199).

ISCModelProperty::DataType Arguments

Here is the signature for the *DataType* function:

```
SC_ValueTypes DataType(VARIANT ValueId)
```

The following table contains the valid arguments for the *DataType* function:

Parameter	Valid Type/Value	Description
ValueId [optional]	Empty	Ignored for scalar properties.
	VT_I4 – index for non-scalar property	Ignored for scalar properties.
	VT_BSTR – name of a member in a non-scalar property	Ignored for scalar properties.

Note: For more information on non-scalar and scalar properties, see the appendix Access CA ERwin PM Properties (see page 199).

ISCModelProperty::RemoveValue Arguments

Here is the signature for the *RemoveValue* function:
VARIANT_BOOL RemoveValue(VARIANT ValueId)

The following table contains the valid arguments for the *RemoveValue* function:

Parameter	Valid Type/Value	Description
ValueId [optional]	Empty	For scalar properties only
	VT_I4 – index of a non-scalar property	Removes the value with the given index in a non-scalar property
	VT_BSTR – name of the property member in a non-scalar property	Removes the value of the non-scalar property member with the given name

ISCModelPropertyCollection

The *ISCModelPropertyCollection* interface is a collection of properties for a given model object. Membership in this collection can be limited by establishing filter criteria.

The following table contains the methods for the *ISCModelPropertyCollection* interface:

Method	Description
ISCModelProperty * Item(VARIANT Class)	Returns a model object property.
long Count()	Number of properties in the collection.
IUnknown _NewEnum()	Constructs an instance of the collection enumerator object.
SC_CLSID * ClassIds()	Returns a SAFEARRAY of property class identifiers in the property collection.
BSTR * ClassNames()	Returns a SAFEARRAY of property type names in the property collection.
SC_ModelPropertyFlags MustBeOn()	Filter on property flags in the collection. The filter is set when the property collection is created through the *ISCModelObject::CollectProperties* method.

Method	Description
SC_ModelPropertyFlags MustBeOff()	Filter on property flags in the collection. The filter is set when the property collection is created through the *ISCModelObject::CollectProperties* method.
VARIANT_BOOL HasProperty(VARIANT ClassId, VARIANT MustBeOn [optional], VARIANT MustBeOff [optional])	Returns TRUE if the object owns a property of the passed class.
ISCModelProperty * Add(VARIANT ClassId)	Construct a new property for a bound model object if it does not exist.
VARIANT_BOOL Remove(VARIANT ClassId)	Removes the indicated property from the bound object.

Note: For information about SC_ModelPropertyFlags, see the Enumerations (see page 107) section.

ISCModelPropertyCollection::Item Arguments

Here is the signature for the *Item* function:

```
ISCModelProperty * Item(VARIANT Class)
```

The following table contains the valid arguments for the *Item* function:

Parameter	Valid Type/Value	Description
Class	VT_BSTR – ID of a property	Returns the property with the given ID.
	VT_BSTR – name of a property	Returns the property with the given name.

Note: For more information about valid property IDs and valid property names, see the Identifiers (see page 172) section.

ISCModelPropertyCollection::HasProperty Arguments

Here is the signature for the *HasProperty* function:

```
VARIANT_BOOL HasProperty(VARIANT ClassId, VARIANT MustBeOn, VARIANT MustBeOff)
```

The following table contains the valid arguments for the *HasProperty* function:

Parameter	Valid Type/Value	Description
ClassId	VT_BSTR – name of a property	Returns TRUE if the property with the given name exists in the collection
	VT_BSTR – ID of a property	Returns TRUE if the property with the given ID exists in the collection
MustBeOn [optional]	VT_I4 – SC_ModelPropertyFlags that must be set	Returns TRUE if the property with the given ClassID has the given property flags set.
	Empty	Default is set to the MustBeOn filter that was used to create the property collection.
MustBeOff [optional]	VT_I4 – SC_ModelPropertyFlags that must not be set	Returns TRUE if the property with the given ClassID does not have the given property flags set.
	Empty	Default is set to the MustBeOff filter that was used to create the property collection.

Note: For information about SC_ModelPropertyFlags, see the Enumerations (see page 107) section.

ISCModelPropertyCollection::Add Arguments

Here is the signature for the *Add* function:

```
ISCModelProperty * Add(VARIANT ClassId)
```

The following table contains the valid arguments for the *Add* function:

Parameter	Valid Type/Value	Description
ClassId	VT_BSTR – name of a property	Adds a new property with the given name.

Parameter	Valid Type/Value	Description
	VT_BSTR – ID of a property	Adds a new property with the given ID.

Note: For more information about valid property names and valid property IDs, see the Identifiers (see page 172) section.

ISCModelPropertyCollection::Remove Arguments

Here is the signature for the *Remove* function:

```
VARIANT_BOOL Remove(VARIANT ClassId)
```

The following table contains the valid arguments for the *Remove* function:

Parameter	Valid Type/Value	Description
ClassId	ISCModelProperty *	Removes the given property.
	VT_BSTR – name of the property	Removes the property with the given name.
	VT_BSTR – ID of the property	Removes the property with the given ID.

Note: For more information about valid property names and valid property IDs, see the Identifiers (see page 172) section.

ISCPersistenceUnit

An *ISCPersistenceUnit* interface is an active model with CA ERwin PM.

The following table contains the methods for the *ISCPersistenceUnit* interface:

Method	Description
BSTR Name()	Passes back a persistence unit name.
SC_OBJID ObjectId()	Passes back an object identifier for the persistence unit.
VARIANT_BOOL DirtyBit()	Returns a flag that indicates that the data has changed in the persistence unit.
void DirtyBit(VARIANT_BOOL)	Sets the flag that indicates that the data in the persistence unit has changed.
ISCModelSet *ModelSet()	Not used in the CA ERwin PM API.

Method	Description
ISCPropertyBag * PropertyBag(VARIANT List [optional], VARIANT AsString [optional])	Returns a property bag with the persistence unit's properties.
void PropertyBag(VARIANT List [optional], VARIANT AsString [optional], ISCPropertyBag * propBag)	Sets a persistence unit with the properties in the given property bag.
VARIANT_BOOL Save(VARIANT Locator [optional], VARIANT Disposition [optional])	Persists model data to external storage.
VARIANT_BOOL HasSession()	Returns TRUE if a unit has one or more sessions connected.
VARIANT_BOOL IsValid()	Returns TRUE if self is valid.

ISCPersistenceUnit::PropertyBag Arguments (Get Function)

Here is the signature for the *PropertyBag (Get)* function:

```
ISCPropertyBag * (VARIANT List, VARIANT AsString)
```

The following table contains the valid arguments for the *PropertyBag (Get)* function:

Parameter	Valid Type/Value	Description
List [optional]	VT_BSTR – semicolon-separated values	Returns a property bag with the unit properties in the given list. The valid values are described in the next table.
AsString [optional]	VT_BOOL	Returns a property bag with all values presented as strings if set to TRUE. Otherwise, the values are presented in native format.

The following table shows the property names and descriptions for property bag members for a persistence unit:

Property Name	Type	Description
Locator	BSTR	Full path of the file that stores the model
Active Model	Boolean	TRUE for the model of which the topmost diagram in the CA ERwin PM application belongs

ISCPersistenceUnit::PropertyBag Arguments (Set Function)

Here is the signature for the *PropertyBag (Set)* function:

```
void PropertyBag(VARIANT List, VARIANT AsString, ISCPropertyBag * propBag)
```

The following table contains the valid arguments for the *PropertyBag (Set)* function:

Parameter	Valid Type/Value	Description
List [optional]	Not used	Not used
AsString [optional]	Not used	Not used

Note: Since the properties of a persistence unit are read-only, this method does not have any functionality.

ISCPersistenceUnit::Save Arguments

Here is the signature for the *Save* function:

```
VARIANT_BOOL Save(VARIANT Locator, VARIANT Disposition)
```

The following table contains the valid arguments for the *Save* function:

Parameter	Valid Type/Value	Description
Locator [optional]	VT_BSTR – full path to a storage location	Saves the model to the specified location
	Empty	Saves the model to the location specified by the *PMFilename* property of the *PMModel* object
Disposition [optional]	VT_BSTR – not used in CA ERwin PM	Not used

ISCPersistenceUnitCollection

The *ISCPersistenceUnitCollection* is a collection of persistence units and is maintained by the *ISCApplication* interface.

The following table contains the methods for the *ISCPersistenceUnitCollection* interface:

Method	Description
ISCPersistenceUnit * Item(VARIANT nIndex)	Passes back an *IUnknown* pointer for a *PersistenceUnit* component identified by its ordered position.
long Count()	Number of persistence units in the collection.
IUnknown _NewEnum()	Constructs an instance of unit enumerator object.
ISCPersistenceUnit * Add(VARIANT Locator, VARIANT Disposition [optional])	Adds a new persistence unit to the unit collection.
VARIANT_BOOL Remove(VARIANT Selector, VARIANT Save [optional])	Removes a persistence unit from the collection.
VARIANT_BOOL Clear()	Purges all units from the collection.
ISCPersistenceUnit * Create(ISCPropertyBag * PropertyBag, VARIANT ObjectId [optional])	Creates a new unit, and registers the unit with the collection.

ISCPersistenceUnitCollection::Add Arguments

Here is the signature for the *Add* function:

```
ISCPersistenceUnit * Add(VARIANT Locator, VARIANT Disposition)
```

The following table contains the valid arguments for the *Add* function:

Parameter	Valid Type/Value	Description
Locator	VT_BSTR – full path to the CA ERwin PM model	Loads the given persistence unit
Disposition [optional]	VT_BSTR – this parameter is not currently being used	Not used

ISCPersistenceUnitCollection::Remove Arguments

Here is the signature for the *Remove* function:

```
VARIANT_BOOL Remove(VARIANT Selector, VARIANT Save)
```

The following table contains the valid arguments for the *Remove* function:

Parameter	Valid Type/Value	Description
Selector	VT_UNKNOWN – pointer to ISCPersistenceUnit interface	Removes the given persistence unit from the collection
	VT_BSTR – ID of a persistence unit	Removes the persistence unit with the given ID from the collection
	VT_I4 – index of a persistence unit in the persistence unit collection	Removes the persistence unit with the given collection index from the collection
Save [optional]	VT_BOOL	If set to TRUE, it saves the persistence unit prior to removing it from the collection

ISCPersistenceUnitCollection::Item Arguments

Here is the signature for the *Item* function:

```
ISCPersistenceUnit * Item(VARIANT nIndex)
```

The following table contains the valid arguments for the *Item* function:

Parameter	Valid Type/Value	Description
nIndex	VT_UNKNOWN – pointer to a session	Retrieves a persistence unit associated with the session
	VT_I4 – index within the collection. Collection index is from 0 to size-1.	Retrieves a persistence unit in the collection with the given index
	VT_BSTR – ID of a persistence unit	Retrieves the persistence unit in the collection with the given ID

ISCPersistenceUnitCollection::Create Arguments

Here is the signature for the *Create* function:

```
ISCPersistenceUnit * Create(ISCPropertyBag * Property Bag, VARIANT ObjectId)
```

The following table contains the valid arguments for the *Create* function:

Parameter	Valid Type/Value	Description
ObjectId [optional]	Empty	CA ERwin PM API assigns an ID to the new persistence unit
	VT_BSTR – object ID for the new persistence unit	CA ERwin PM API assigns a given ID to the new persistence unit

The following table describes properties that are set in an instance of *ISCPropertyBag* when creating a new persistence unit:

Property	Type	Value
Name	BSTR	Name of the new model
ModelType	BSTR	■ IDEF0 (this is the default if a value is not provided) ■ DFD ■ IDEF3
	VT_I2	■ 0 – IDEF0 (this is the default if a value is not provided) ■ 1 – DFD ■ 2 – IDEF3
UseBlankModel	boolean	TRUE if you create a model without default objects. FALSE if you create a model with default objects. FALSE is the default value.

The *ModelType* property determines what type of context activity to create when the model is created.

ISCPropertyBag

The *ISCPropertyBag* interface is used to set and access the properties of *ISCApplicationEnvironment*. The *ISCPropertyBag* is also used to set the properties of a new persistence unit.

The following table contains the methods for the *ISCPropertyBag* interface:

Method	Description
long Count()	Returns the number of properties
VARIANT Value(VARIANT Property)	Retrieves the indicated property in the bag
void Value(VARIANT Property, VARIANT Val)	Sets the indicated property in the bag
BSTR Name(long PropertyIdx)	Retrieves the indicated property name in the bag
void ClearAll()	Removes all properties from the bag
VARIANT_BOOL Add(BSTR Name, VARIANT Value)	Adds a new property to the bag

ISCPropertyBag::Value Arguments (Get Function)

Here is the signature for the *Value (Get)* function:

VARIANT Value(VARIANT Property)

The following table contains the valid arguments for the *Value (Get)* function:

Parameter	Valid Type/Value	Description
Property	VT_BSTR – name of the property	Returns the value of the property with the given name in the property bag
	VT_I4 – index of the property	Returns the value of the property with the given index in the property bag

ISCPropertyBag::Value Arguments (Set Function)

Here is the signature for the *Value (Set)* function:
void Value(VARIANT Property, VARIANT Val)

The following table contains the valid arguments for the *Value (Set)* function:

Parameter	Valid Type/Value	Description
Property	VT_BSTR – name of the property	Sets the property with the given name with the given value
	VT_BSTR – ID of the property	Sets the property with the given ID with the given value
Val	Dependent on property	Value for the given property

ISCPropertyBag::Add Arguments

Here is the signature for the *Add* function:
VARIANT_BOOL Add(BSTR Name, VARIANT Value)

The following table contains the valid arguments for the *Add* function:

Parameter	Valid Type/Value	Description
Value	All VARIANTs	Returns TRUE if the property was added to the bag, otherwise, it is FALSE

ISCPropertyValue

The *ISCPropertyValue* interface is a single value of a given property.

The following table contains the methods for the *ISCPropertyValue* interface:

Method	Description
VARIANT ValueId(VARIANT ValueType [optional])	Uniquely identifies the value in a non-scalar property.
SC_CLSID PropertyClassId()	Returns the class identifier of the current property.
BSTR PropertyClassName()	Returns the class name of the current property.
VARIANT Value(VARIANT ValueType [optional])	Converts the current value to the passed value type.

Method	Description
SC_ValueTypes ValueType()	Passes back the default type of the property value.
SC_ValueTypes ValueIdType()	Passes back the default type of the ValueId that identifies the value within the non-scalar property.
SC_ValueTypes * GetSupportedValueTypes()	Groups a list of supported value types and returns it as a SAFEARRAY.
SC_ValueTypes * GetSupportedValueIdTypes()	Groups a list of supported value types for the current value identifier and return it as a SAFEARRAY.

Note: For information on non-scalar property value IDs, see the appendix Access CA ERwin PM Properties (see page 199).

ISCPropertyValue::ValueId Arguments

Here is the signature for the *ValueId* function:

```
VARIANT ValueId(VARIANT ValueType)
```

The following table contains the valid arguments for the *ValueId* function:

Parameter	Valid Type/Value	Description
ValueType [optional]	VT_I4 – SCVT_I2	Returns VT_EMPTY if property is scalar. If non-scalar, the value of the property index is returned.
	VT_I4 – SCVT_I4	Returns VT_EMPTY if property is scalar. If non-scalar, the value of the property index is returned.
	VT_I4 – SCVT_BSTR	Returns VT_EMPTY if property is scalar. Name of the non-scalar property member if available or else it is the index of the member.
	VT_I4 – SCVT_DEFAULT	Returns VT_EMPTY if property is scalar. If non-scalar, the value of the property index is returned.
	Empty	Defaults to SCVT_Default.

Note: For information on non-scalar property value IDs, see the appendix Access CA ERwin PM Properties (see page 199).

ISCPropertyValue::Value Arguments

Here is the signature for the *Value* function:

```
VARIANT Value(VARIANT ValueType)
```

The following table contains the valid arguments for the *Value* function:

Parameter	Valid Type/Value	Description
ValueType [optional]	VT_I4 – SCVT_DEFAULT	See note below
	VT_I4 – SCVT_BSTR	String representation of the property value.
	VT_I4 – type of property	See note below
	Empty	Defaults to SCVT_DEFAULT.

Note: For information on the returned value of a given property, see the appendix Access CA ERwin PM Properties (see page 199).

ISCPropertyValueCollection

The *ISCPropertyValueCollection* interface is a collection of values for a non-scalar property.

The following table contains the methods for the *ISCPropertyValueCollection* interface:

Method	Description
ISCPropertyValue * Item(VARIANT ValueId)	Returns a single value from the property value collection
long Count()	Number of values in the collection
IUnknown _NewEnum()	Constructs an instance of the collection enumerator object

ISCPropertyValueCollection::Item Arguments

Here is the signature for the *Item* function:

```
ISCPropertyValue * Item(VARIANT ValueId)
```

The following table contains the valid arguments for the *Item* function:

Parameter	Valid Type/Value	Description
ValueId	VT_I2 – index of the member in non-scalar property	See note below
	VT_BSTR – name of a member in non-scalar property	See note below

Note: For information on the return value, see the appendix Access CA ERwin PM Properties (see page 199).

ISCSession

The *ISCSession* interface is an active connection between the API client and a model. In order to access models, API clients create sessions and open them against persistence units.

The following table contains the methods for the *ISCSession* interface:

Method	Description
BSTR Name()	Name of the associated persistence unit.
ISCModelObjectCollection * ModelObjects()	Creates a *ModelObject* collection for the session.
ISCPersistenceUnit * PersistenceUnit()	Persistence unit associated with the session.
SC_SessionLevel Level()	Returns the level at which the persistence unit is bound.
SC_SessionFlags Flags()	Returns a set of flags associated with session.
VARIANT_BOOL Close()	Disconnects self from its associated persistence unit.
VARIANT_BOOL Open(ISCPersistenceUnit * Unit, VARIANT Level [optional], VARIANT Flags [optional])	Binds self to the persistence unit identified by the *Unit* parameter.
VARIANT_BOOL IsOpen()	TRUE if and only if the session is open.
VARIANT_BOOL ChangeAccess(SC_SessionFlags Flags)	Changes the model access to the specified level.

Method	Description
VARIANT BeginTransaction()	Opens a transaction on the session.
VARIANT BeginNamedTransaction(BSTR name)	Opens a transaction on the session with the given name.
VARIANT_BOOL CommitTransaction(VARIANT TransactionId)	Commits the specified transaction.
VARIANT_BOOL IsValid()	Returns TRUE if self is valid.
VARIANT_BOOL IsTransactionEmpty()	Returns TRUE if there are no transactions that are active.
VARIANT_BOOL RollbackTransaction(VARIANT TransactionId)	Returns the depth of the nested transactions. **Note:** The CA ERwin PM API does not support rollback of transactions so this function should not be used.
long TransactionDepth()	Returns the depth of the nested transactions. **Note:** The CA ERwin PM API does not support nested transactions so this function should not be used.
SCOBJID ModelSetId()	Not used in CA ERwin PM API.

Note: For more information on SC_SessionLevel and SC_SessionFlags, see the Enumerations (see page 107) section.

ISCSession::Open Arguments

Here is the signature for the *Open* function:

VARIANT_BOOL Open(ISCPersistenceUnit * Unit, VARIANT Level, VARIANT Flags)

The following table contains the valid arguments for the *Open* function:

Parameter	Valid Type/Value	Description
Unit	Pointer to a persistence unit that was loaded	Attaches the persistence unit to the session
	Pointer to a persistence unit variable but there is no persistence unit loaded	This is only valid during metamodel access
Level [optional]	Empty	Defaults to SCD_SL_M0
	SCD_SL_M0	Data-level access
	SCD_SL_M1	Metamodel access

Parameter	Valid Type/Value	Description
Flags [optional]	Empty	Defaults to SCD_SF_NONE
	SCD_SF_NONE	Other sessions can have access to the attached persistence unit
	SCD_SF_EXCLUSIVE	Other sessions cannot have access to the attached persistence unit

ISCSession::CommitTransaction Arguments

Here is the signature for the *CommitTransaction* function:

```
VARIANT_BOOL CommitTransaction(VARIANT TransactionId)
```

The following table contains the valid arguments for the *CommitTransaction* function:

Parameter	Valid Type/Value	Description
TransactionId	The ID of the session	Commits the active transaction of the session

ISCSessionCollection

The *ISCSessionCollection* interface is a collection of sessions and is maintained by the *ISCApplication* interface.

The following table contains the methods for the *ISCSessionCollection* interface:

Method	Description
ISCSession * Item(long nIndex)	Passes back a session identified by its ordered position
long Count()	Number of sessions in the collection
IUnknown _NewEnum()	Constructs an instance of *Session* enumerator object
ISCSession * Add()	Construct a new, closed *Session* object, and adds it to the collection
VARIANT_BOOL Remove(VARIANT SessionId)	Removes a *Session* object from the collection
VARIANT_BOOL Clear()	Removes all *Session* objects from the collection

ISCSessionCollection::Remove Arguments

Here is the signature for the *Remove* function:
VARIANT_BOOL Remove(VARIANT SessionId)

The following table contains the valid arguments for the *Remove* function:

Parameter	Valid Type/Value	Description
SessionId	VT_UNKNOWN – pointer to the ISCSession interface	Removes the given session from the collection
	VT_I4 – index in the session collection	Removes the session with the given index from the collection

Enumerations

This section contains information regarding the various enumerations for the API. The enumerations define valid values for various properties.

errorERwin

The following table contains the properties and enumerations for *errorERwin*:

Property	Enumeration
SCAPI_ERWIN_BAG_KEY	-2147219462
SCAPI_ERWIN_BAG_VALUE	-2147219461
SCAPI_ERWIN_BAG_CATEGORY	-2147219431
SCAPI_ERWIN_BAG_CATEGORY_PROPERTY	-2147219430
SCAPI_ERWIN_UNIT_REMOVE	-2147219456
SCAPI_ERWIN_BAG_SASD	-2147219438
SCAPI_ERWIN_LEVEL	-2147219486
SCAPI_ERWIN_SESSION_EXCLUSIVE	-2147219482
SCAPI_ERWIN_LEVEL_TRANSACTION	-2147219481
SCAPI_ERWIN_SESSION_NOTOPEN	-2147219439
SCAPI_ERWIN_TRANSACTION_FATAL	-2147219485
SCAPI_ERWIN_TRANSACTION_ERROR	-2147219484

Property	Enumeration
SCAPI_ERWIN_TRANSACTION_WARNING	-2147219483
SCAPI_ERWIN_ROLLBACK_FATAL	-2147219480
SCAPI_ERWIN_ROLLBACK_ERROR	-2147219479
SCAPI_ERWIN_ROLLBACK_WARNING	-2147219478
SCAPI_ERWIN_ACTIVE_TRANSACTION	-2147219477
SCAPI_ERWIN_UNCOMM_TRANSACTION	-2147219441
SCAPI_ERWIN_NOPROVIDER	-2147219499
SCAPI_ERWIN_NOLOCATOR	-2147219498
SCAPI_ERWIN_NODISPOSITION	-2147219497
SCAPI_ERWIN_MISPLACED_KEY	-2147219453
SCAPI_ERWIN_FILE_SAVE_NEW	-2147219452
SCAPI_ERWIN_FILE_EXT	-2147219495
SCAPI_ERWIN_FILE_OPEN	-2147219494
SCAPI_ERWIN_FILE_OPEN_EXISTS	-2147219445
SCAPI_ERWIN_FILE_EXISTS	-2147219493
SCAPI_ERWIN_FILE_READONLY	-2147219492
SCAPI_ERWIN_FILE_NOTEXISTS	-2147219491
SCAPI_ERWIN_FILE_READ	-2147219490
SCAPI_ERWIN_FILE_SAVE	-2147219455
SCAPI_ERWIN_FILE_REMOVE	-2147219454
SCAPI_ERWIN_FILE_SAVE_READONLY	-2147219451
SCAPI_ERWIN_FILE_SAVE_EXT	-2147219450
SCAPI_ERWIN_FILE_SAVE_ACCESS	-2147219449
SCAPI_ERWIN_FILE_SAVE_EXISTS	-2147219448
SCAPI_ERWIN_FILE_SAVE_FORMAT	-2147219446
SCAPI_ERWIN_FILE_VERSION	-2147219434
SCAPI_ERWIN_FILE_VERSION1	-2147219433
SCAPI_ERWIN_FILE_VERSION2	-2147219433
SCAPI_ERWIN_FILE_SEARCH_ERROR	-2147219435
SCAPI_ERWIN_MMART_SERVER_TYPE	-2147219404

Property	Enumeration
SCAPI_ERWIN_MMART_DATABASE	-2147219403
SCAPI_ERWIN_MMART_SERVER_ATTR	-2147219402
SCAPI_ERWIN_MMART_SERVER_CONN	-2147219401
SCAPI_ERWIN_MMART_SERVER_DCONN	-2147219379
SCAPI_ERWIN_MMART_REPOSITORY_CRT	-2147219400
SCAPI_ERWIN_MMART_SESSION_LOCK	-2147219399
SCAPI_ERWIN_MMART_SESSION_OPEN	-2147219398
SCAPI_ERWIN_MMART_MODEL_OPEN	-2147219397
SCAPI_ERWIN_MMART_UI	-2147219396
SCAPI_ERWIN_MMART_DISP	-2147219395
SCAPI_ERWIN_MMART_CONNECTION	-2147219383
SCAPI_ERWIN_MMART_VERSION	-2147219382
SCAPI_ERWIN_MMART_TERMINATE	-2147219381
SCAPI_ERWIN_MMART_SAVE_NEW	-2147219394
SCAPI_ERWIN_MMART_SAVE_EXISTS	-2147219393
SCAPI_ERWIN_MMART_SAVE_EXISTS1	-2147219392
SCAPI_ERWIN_MMART_SAVE_ACCESS	-2147219391
SCAPI_ERWIN_MMART_SAVE_LIBRARY	-2147219390
SCAPI_ERWIN_MMART_SAVE_ERROR	-2147219389
SCAPI_ERWIN_MMART_SAVE_CONNECTION	-2147219388
SCAPI_ERWIN_MMART_SAVE_SUBMODEL	-2147219387
SCAPI_ERWIN_MMART_SAVE_VERSION	-2147219386
SCAPI_ERWIN_MMART_SAVE_READONLY	-2147219385
SCAPI_ERWIN_MMART_SAVE_SERVERCHANGE	-2147219384
SCAPI_ERWIN_NOMODEL	-2147219503
SCAPI_ERWIN_MODEL_ID	-2147219447
SCAPI_ERWIN_CLASS	-2147219488
SCAPI_ERWIN_CLASSNAME	-2147219487
SCAPI_ERWIN_ID_TAKEN	-2147219476
SCAPI_ERWIN_CLASS1	-2147219475

Property	Enumeration
SCAPI_ERWIN_REMOVE_TYPE	-2147219474
SCAPI_ERWIN_NOOBJECT	-2147219502
SCAPI_ERWIN_NOPROPERTY	-2147219501
SCAPI_ERWIN_ADD_PROPERTY	-2147219473
SCAPI_ERWIN_PROPERTY_CLASS	-2147219436
SCAPI_ERWIN_PROPERTY_CLASSNAME	-2147219437
SCAPI_ERWIN_POPULATE_COLLECTION	-2147219444
SCAPI_ERWIN_POPULATE_VARIANT	-2147219443
SCAPI_ERWIN_POPULATE_SCALAR	-2147219442
SCAPI_ERWIN_NOCONVERSION	-2147219500
SCAPI_ERWIN_VALUEID_NOCONVERSION	-2147219440
SCAPI_ERWIN_SAFEARRAY	-2147219428
SCAPI_ERWIN_PROPERTY_RETIRED	264219

errorSCAPI

The following table contains the properties and enumerations for *errorSCAPI*:

Property	Enumeration
SCAPI_E_POINTER	-2147220501
SCAPI_E_CREATEFAILED	-2147220500
SCAPI_E_QIFAILED	-2147220499
SCAPI_E_CALLFAILED	-2147220493
SCAPI_E_APPLICATION	-2147220492
SCAPI_E_OUTOFMEMORY	-2147220495
SCAPI_E_INVALIDARG	-2147220498
SCAPI_E_MISSINGARG	-2147220497
SCAPI_E_POINTERARG	-2147220496
SCAPI_E_INDEXARG	-2147220494
SCAPI_E_ARGRANGE	-2147220491
SCAPI_E_UC_SESSION	-2147220454

Property	Enumeration
SCAPI_E_UC_CLEAR_SESSION	-2147220453
SCAPI_E_UC_CLEAR	-2147220452
SCAPI_E_UC_NOTINCOLLECTION	-2147220451
SCAPI_E_UC_ADDPERSISTENCEUNIT	-2147220450
SCAPI_E_UC_CREATEPERSISTENCEUNIT	-2147220449
SCAPI_E_MDC_ADDDIRECTORY	-2147220204
SCAPI_E_MDC_DIRECTORYOBJ	-2147220203
SCAPI_E_SC_SESSIONOBJ	-2147220444
SCAPI_E_SC_ADDSESSION	-2147220439
SCAPI_E_SS_NOTINCOLLECTION	-2147220443
SCAPI_E_SS_OPEN	-2147220442
SCAPI_E_SS_NOTOPEN	-2147220441
SCAPI_E_SS_TRANSACTION	-2147220440
SCAPI_E_MOC_OBJECT	-2147220404
SCAPI_E_MOC_ID	-2147220403
SCAPI_E_MOC_CLASS	-2147220402
SCAPI_E_MOC_CLASSNAME	-2147220401
SCAPI_E_MOC_CLASSNAME1	-2147220301
SCAPI_E_MOC_PROPERTY	-2147220304
SCAPI_E_MOC_PROPERTY1	-2147220303

SC_ModelObjectFlags

The following table contains the properties and enumerations for *SC_ModelObjectFlags*:

Property	Flag Bit	Enumeration	Description
SCD_MOF_DONT_CARE		0	No flags are set
SCD_MOF_PERSISTENCE_UNIT	0	1	Object is a persistence unit (for example, model)
SCD_MOF_USER_DEFINED	1	2	Object is user-defined (for example, user-defined properties)

Property	Flag Bit	Enumeration	Description
SCD_MOF_ROOT	2	4	Object is the root object (for example, model)
SCD_MOF_TOOL	3	8	Object is maintained by the tool (however, not used in CA ERwin PM)
SCD_MOF_DEFAULT	4	16	Object is created by the tool and not removable (however, not used in CA ERwin PM)
SCD_MOF_TRANSACTION	5	32	Object is new or updated in a transaction and the transaction was not committed

SC_ModelPropertyFlags

The following table contains the properties and enumerations for *SC_ModelPropertyFlags*:

Property	Flag Bit	Enumeration	Description
SCD_MPF_DONT_CARE		0	No flags are set
SCD_MPF_NULL	0	1	Property has NULL value or no value
SCD_MPF_USER_DEFINED	1	2	Property is user-defined
SCD_MPF_SCALAR	2	4	Property is scalar
SCD_MPF_TOOL	3	8	Property is maintained by the tool (however, not used in CA ERwin PM)
SCD_MPF_READ_ONLY	4	16	Property is read-only (however, not used in CA ERwin PM)
SCD_MPF_DERIVED	5	32	Property is inherited, calculated, or derived (however, not used in CA ERwin PM)
SCD_MPF_OPTIONAL	6	64	Property is optional and can be removed

SC_SessionFlags

The following table contains the properties and enumerations for *SC_SessionFlags*:

Property	Enumeration	Description
SCD_SF_NONE	0	Session has non-exclusive access to its connected persistence unit. Other sessions can connect to the same persistence unit.
SCD_SF_EXCLUSIVE	1	Session has exclusive access to its connected persistence unit. No other sessions are allowed to access the persistence unit.

SC_SessionLevel

The following table contains the properties and enumerations for *SC_SessionLevel*:

Property	Enumeration	Description
SCD_SL_NONE	-1	Not used
SCD_SL_M0	0	Data level access
SCD_SL_M1	1	Metamodel access
SCD_SL_MM	2 (not used in CA ERwin PM)	Not used
SCD_SL_MD0	3 (not used in CA ERwin PM)	Not used
SCD_SL_MAX	4 (not used in CA ERwin PM)	Not used

SC_ValueTypes

The following table contains the properties and enumerations for *SC_ValueTypes*:

Property	Enumeration	Description
SCVT_NULL	0	Missing value
SCVT_I2	1	Signed 16-bit integer

Property	Enumeration	Description
SCVT_I4	2	Signed 32-bit integer
SCVT_UI1	3	Unsigned 8-bit integer. Do not use this type to hold character data.
SCVT_R4	4	4 byte floating point real
SCVT_R8	5	8 byte floating point real
SCVT_BOOLEAN	6	Boolean
SCVT_CURRENCY	7	64-bit currency value
SCVT_IUNKNOWN	8	IUnknown interface pointer
SCVT_IDISPATCH	9	IDispatch interface pointer
SCVT_DATE	10	Date value in VARIANTDATE format
SCVT_BSTR	11	String
SCVT_UI2	12	Unsigned 16-bit integer
SCVT_UI4	13	Unsigned 32-bit integer
SCVT_GUID	14	GUID
SCVT_OBJID	15	Object identifier with offset
SCVT_BLOB	16	SAFEARRAY of unsigned BYTEs
SCVT_DEFAULT	17	Default value type
SCVT_I1	18	Signed 1 byte integer. Do not use this type to hold character data.
SCVT_INT	19	Machine-dependent signed integer
SCVT_UINT	20	Machine-dependent unsigned integer
SCVT_RECT	21	Rectangle-array of four integers.
SCVT_POINT	22	Point-array of two integers

Appendix B: CA ERwin PM Metamodel

This appendix describes the CA ERwin PM metamodel. The object hierarchy is shown, followed by tables of valid properties by object.

This section contains the following topics:

Object Hierarchy

The following illustrates the object hierarchy of the CA ERwin PM metamodel:

PMModel
- PMActivity
  - PMActivityCost
  - PMUDPInstance
    - PMUDPValue
- PMArrow
  - PMUDPInstance
    - PMUDPValue
  - PMAssociation
    - PMIRUN
    - PMCRUD
- PMBitmap
- PMCostCenter
- PMDataStore
  - PMUDPInstance
    - PMUDPValue
- PMDiagram
  - PMArrowLabel
    - PMUDPInstance (Swim Lanes only)
      - PMUDPValue
    - PMArrowSegment
  - PMBorder
  - PMBox
  - PMNode
  - PMRuler (Swim Lanes only)
  - PMTextBlock
  - PMTunnel

PMERwinEntity
  PMERwinAttribute
PMERwinModel
PMExternal
  PMUDPInstance
    PMUDPValue
PMFont
PMJunction
  PMUDPInstance
    PMUDPValue
PMNodeTree
  PMOrgChartBox
  PMOrgChartRow
  PMTextBlock
PMPalette
  PMColor
PMResource
PMReferent
  PMUDPInstance
    PMUDPValue
PMRole
PMRoleGroup
PMRRGAssociation
PMUDPCategory
PMUDPDefinition
  PMUDPValue

Valid Properties by Object

The tables in the following sections describe valid properties for each CA ERwin PM metamodel object. In some cases, special instructions on the format variations are defined.

Datatype Definitions

The following table shows the datatypes and their definitions:

Datatype	Enumeration Value	Description
DT_Invalid	0	invalid datatype
DT_char	1	signed character
DT_short	2	short
DT_long	3	long

Datatype	Enumeration Value	Description
DT_enum	4	enum
DT_double	5	double, float, real
DT_LPSTR	6	pointer to a string (or other data)
DT_LPNewStr	7	pointer to a string (or other data) that client must free
DT_Cstring	8	pointer to a *CString* object
DT_NewCString	9	pointer to a *CString* object that client must delete
DT_LPLOGFONT	10	pointer to a LOGFONT structure
DT_BOOL	11	boolean value
DT_ref	12	pointer to a *PMBase* object
DT_LPPMID	13	pointer to an *OriginId*
DT_Byte	14	unsigned character
DT_sDate	15	PM date structure
DT_POINT	16	coordinate point (long, long)
DT_RECT	17	rectangle (long, long, long, long)
DT_sSelList	18	multi-select list
DT_sCharArray	19	character array header structure
DT_sShortArray	20	short array header structure
DT_sLongArray	21	long array header structure
DT_sDoubleArray	22	double array header structure
DT_sStrArray	23	string array header structure
DT_sByteArray	24	byte array header structure
DT_sPointArray	25	coordinate array header structure
DT_sRefArray	26	base object pointer array header structure

PMApplication Object

The class is *PMApplication*. The following table describes the optional properties for this object:

Identifier	Datatype	Description
PM_IDL_AutoSpaceArrows	BOOL	IDL Import Auto-space Arrows
PM_IDL_ExportCostData	BOOL	IDL Export Cost Data
PM_IDL_ExportDataUsage	BOOL	IDL Export Data Usage
PM_IDL_ExportUDPs	BOOL	IDL Export UDPs
PM_IDL_ImportMode	DT_long	IDL Import Mode: 0 – Standard IDL 1 – AIOwin variation 2 – Meta variation 3 – Wizdom variation
PM_OrgChartDelimiter	string	Organization Chart Delimiter
PM_ShowJunctNumbers	BOOL	Display Junction Numbers
PM_ShowUDPMarker	BOOL	Display UDP Marker
PM_ShowUOWProps	BOOL	Display UOW Properties
PM_UseBPwinColors	BOOL	Use BPwin Colors
PM_WinsOnLoad	BOOL	Create windows on model load

PMNewModelDefaults Object

The class is *PMNewModelDefaults*. The following table describes the optional properties for this object:

Identifier	Datatype	Description
PM_Inch_OR_MM	BOOL	0 – Imperial (English) units 1 – Metric units
PM_SheetSize	DT_long	SheetSize: 0 – Custom; must set offsets, diagram area, and printable area 1 – 11” x 8.5” A size Landscape 2 – 8.5” x 11” A size Portrait 3 – 14” x 8.5” Legal Landscape 4 – 8.5” x 14” Legal Portrait 5 – 17” x 11” B size Landscape 6 – 11” x 17” B size Portrait 7 – 22” x 17” C size Landscape 8 – 17” x 22” C size Portrait 9 – A4: 297mm x 210mm Landscape
		10 – A4: 210mm x 297mm Portrait 11 – A3: 420mm x 297mm Landscape 12 – A3: 297mm x 420mm Portrait 13 – A2: 594mm x 420mm Landscape 14 – A2: 420mm x 594mm Portrait 15 – A1: 840mm x 594mm Landscape 16 – A1: 594mm x 840mm Portrait

Identifier	Datatype	Description
PMActBoxCornerDisplayType	DT_long	ABC Units: 0 – Cost 1 – Frequency 2 – Duration
PMActiveAreaDimension	POINT	Diagram Area (Width,Height)
PMActivityNumberConventionFlag	DT_long	Activity Numbering Convention: 0 – Single (for example, 1, 2, 3, …) 1 – Use Diagram numbering convention 2 – None
PMAllowBoxMovementFlag	Byte	0 – Do not allow boxes to be moved 1 – Allows boxes to be moved
PMAllowBoxResizeFlag	Byte	0 – Do not allow boxes to be resized 1 – Allow boxes to be resized
PMAuthor	string	Author
PMAuthorInitials	string	Author Initials
PMAutoSpaceArrowsFlag	Byte	0 – Do not auto space arrows 1 – Auto space arrows
PMCostDecimalPlaces	DT_Byte	Currency decimals in diagrams (upper 4 bits) and in reports (lower 4 bits)
PMCostUnit	string	Currency Description
PMCurrencyPlacement	DT_long	Symbol Placement: 1 – Left (for example, "$1") 2 – Right (for example, "1$") 3 – LeftWithSpace (for example, "$ 1") 4 – RightWithSpace (for example, "1 $")
PMCurrencySymbol	string	Currency Symbol

Identifier	Datatype	Description
PMDataUsageRules	Byte	0 – Do not enforce data usage rules 1 – Enforce data usage rules
PMDiagramNumberConventionFlag	DT_long	Diagram Numbering Convention: 0 – NoDots (for example, 1, 11, 111) 1 – UseDots (for example, 1, 1.1, 1.1.1)
PMDimension	POINT	Printable Area (Width,Height)
PMDisplayBoxHighlightingFlag	BOOL	Display Block Highlighting
PMDisplayColorsFlag	Byte	0 – Do not display colors 1 – Display colors
PMDisplayLeafCornersFlag	Byte	0 – Do not display leaf flag on undecomposed boxes 1 – Display leaf flag on undecomposed boxes
PMDisplayShadowsFlag	Byte	0 – Do not display box shadows 1 – Display box shadows
PMDisplaySquigglesFlag	Byte	0 – Do not display squiggles 1 – Display squiggles
PMDurationDecimalPlaces	long	Time decimals in duration values
PMFitActivityTextInBoxFlag	DT_long	Fit Names in Boxes: 0 – Do not resize or wrap 1 – Wrap text to fit box 2 – Automatically resize box to fit text
PMFrequencyDecimalPlaces	long	Time decimals in frequency values

Identifier	Datatype	Description
PMHdrFtrType0	DT_long	Left Header: 0 – None 1 – Author 2 – Author Initials 3 – C-Number 4 – CreationDate 5 – Current Date (long format) 6 – Current Date (short format) 7 – Diagram Name 8 – Diagram Number 9 – Diagram Status 10 – Model Name 11 – Model Status 12 – Page Number 13 – Project Name 14 – Revision Date
PMHdrFtrType1	DT_long	Center Header See the PMHdrFtrType0 property for the enumeration values
PMHdrFtrType2	DT_long	Right Header See the PMHdrFtrType0 property for the enumeration values
PMHdrFtrType3	DT_long	Left Footer See the PMHdrFtrType0 property for the enumeration values
PMHdrFtrType4	DT_long	Center Footer See the PMHdrFtrType0 property for the enumeration values

Identifier	Datatype	Description
PMHdrFtrType5	DT_long	Right Footer See the PMHdrFtrType0 property for the enumeration values
PMLongDateFlag	Byte	0 – Do not display dates in long format 1 – Display dates in long format
PMNextBoxNumber	long	Next UOW Number
PMNextDatastoreNumber	long	Next Data Store Number
PMNextExternalNumber	long	Next External Number
PMNoAutoRenumberFlag	BOOL	Use Persistent Activity Numbers
PMOffpageRefIdFlag	DT_long	Off Page Reference Label: 0 – C-Number 1 – Node Number 2 – Diagram Name
PMOriginOffset	POINT	PageOffset (x,y)
PMPrefix	long	Activity Number Prefix
PMShowActivityCostFlag	BOOL	Display ABC Data
PMShowActivityNumberingFlag	BOOL	Display Activity Numbers
PMShowActivityPrefixFlag	BOOL	Show Activity Prefix
PMShowArrowBreakFlag	DT_long	Break Arrows at Intersections: 0 – Do not break arrows 1 – Break Horizontal Arrows 2 – Break Vertical Arrows
PMShowArrowNameFlag	BOOL	Display Arrow Names
PMShowDatastoreNumberingFlag	BOOL	Display Data Store Numbers
PMShowDatastorePrefixFlag	BOOL	Show Prefix on Data Stores
PMShowDiagramPrefixFlag	BOOL	Show Diagram Prefix
PMShowExternalNumberingFlag	BOOL	Display External Numbers
PMShowExternalPrefixFlag	BOOL	Show Prefix on Externals

Identifier	Datatype	Description
PMShowICOMCodeFlag	Byte	0 – Do not show ICOM codes 1 – Show ICOM codes
PMShowIDEF0KitFlag	DT_long	Header Type: 0 – No header 1 – Use IDEF0 kit 2 – Use custom header
PMShowIDEF0TitleFlag	DT_long	Footer Type: 0 – No footer 1 – Use IDEF0 title 2 – Use custom footer
PMShowTunnelFlag	BOOL	Display Tunnels
PMSortArrowsFlag	BOOL	Sort Arrows (IDEF3 and DFD only)
PMTimeUnit	string	Time Unit

PMModel Object

The following table describes the required properties for the *PMModel* object:

Property Name	Datatype	Description
Name	DT_LPSTR	Model Name

The following table describes optional properties for the *PMModel* object:

Property Name	Datatype	Description
PMActBoxCornerDisplayType	DT_long	ABC Units: 0 – Cost 1 – Frequency 2 – Duration
PMActiveAreaDimension	DT_POINT	Diagram Area (Width,Height)

Property Name	Datatype	Description
PMActivityNumberConventionFlag	DT_long	Activity Numbering Convention: 0 – Single (for example, 1, 2, 3 …) 1 – Use Diagram numbering convention 2 – None
PMAllowBoxMovementFlag	DT_Byte	0 – Do not allow boxes to be moved 1 – Allow boxes to be moved
PMAllowBoxResizeFlag	DT_Byte	0 – Do not allow boxes to be resized 1 – Allow Boxes to be resized
PMAPIMod	DT_BOOL	0 – Model was not modified using the API or XML 1 – Model was modified using the API or XML This value cannot be manually reset
PMArrowNameColorReference	DT_ref	Default Arrow Color reference (*PMColor*)
PMArrowTextColorReference	DT_ref	Default Arrow Name Color reference (*PMColor*)
PMAuthor	DT_LPSTR	Author
PMAuthorInitials	DT_LPSTR	Author Initials
PMAutoSpaceArrowsFlag	DT_Byte	0 – Do not autospace arrows 1 – Autospace arrows
PMBitmapJustification	DT_long	0 – Left 1 – Center 2 – Right
PMBitmapRefList[0]	DT_ref	IDEF0 Activity Box Bitmap
PMBitmapRefList[1]	DT_ref	DFD Activity Box Bitmap
PMBitmapRefList[2]	DT_ref	IDEF3 Activity Box Bitmap
PMBitmapRefList[3]	DT_ref	Data Store Box Bitmap
PMBitmapRefList[4]	DT_ref	External Box Bitmap
PMBitmapRefList[5]	DT_ref	Referent Box Bitmap
PMBitmapRefList[6]	DT_ref	Asynchronous AND Junction Box Bitmap

Property Name	Datatype	Description
PMBitmapRefList[7]	DT_ref	Synchronous AND Junction Box Bitmap
PMBitmapRefList[8]	DT_ref	Asynchronous OR Junction Box Bitmap
PMBitmapRefList[9]	DT_ref	Synchronous OR Junction Box Bitmap
PMBitmapRefList[10]	DT_ref	XOR Junction Box Bitmap
PMBorderSystemDefaultFontRef	DT_ref	Font used for rotated IDEF0 style external references
PMBorderSystemFontRef	DT_ref	Border system text font reference (*PMFont*)
PMBorderUserFontRef	DT_ref	Border user text reference (*PMFont*)
PMBoxBackgroundColorReference	DT_ref	Default Box Background color reference (*PMColor*)
PMBPXPath	DT_LPSTR	Path of BPX file to read for update
PMBPXSync4	DT_BOOL	0 – Model was not synchronized with a CA ERwin DM Version 4.0 or later model using BPX 1 – Model was synchronized with a CA ERwin DM Version 4.0 or later model using BPX
PMContextDiagramActivityFontRef	DT_ref	Default Context Diagram activity font
PMContextDiagramArrowFontRef	DT_ref	Default Context Diagram arrow font
PMCostDecimalPlaces	DT_Byte	Currency decimals in diagrams (upper 4 bits) and in reports (lower 4 bits)
PMCostUnit	DT_LPSTR	Currency Description
PMCreatedDate	DT_sDate	Creation Date
PMCreationBuild	DT_long (Private)	Build number revision at which model was created *HIWORD* of *nCreationDataRev*
PMCreationDataRev	DT_long (Private)	Data revision at which model was created (0 for data revisions less than 57) *LOWORD* of *nCreationDataRev*

Property Name	Datatype	Description
PMCurrencyPlacement	DT_long	Symbol Placement: 1 – Left (for example, ”$1”) 2 – Right (for example, ”1$”) 3 – LeftWithSpace (for example, ”$ 1”) 4 – RightWithSpace (for example, ”1 $”)
PMCurrencySymbol	DT_LPSTR	Currency Symbol
PMDataUsageRules	DT_Byte	0 – If data usage rules should not be applied to CRUD/IRUN 1 – If data usage rules should be applied to CRUD/IRUN
PMDecompDiagramActivityFontRef	DT_ref	Default Decomposition Diagram activity font
PMDecompDiagramArrowFontRef	DT_ref	Default Decomposition Diagram arrow font
PMDescriptionString	DT_LPSTR	Definition
PMDiagramFontRef	DT_ref	Parent Diagram font reference (*PMFont*)
PMDiagramNumberConventionFlag	DT_long	Diagram Numbering Convention: 0 – NoDots (for example, 1, 11, 111) 1 – UseDots (for example, 1, 1.1, 1.1.1)
PMDiagramTitleColorReference	DT_ref	Default Diagram Title Color reference (*PMColor*)
PMDiagramTitleFontRef	DT_ref	Parent Diagram title font reference (*PMFont*)
PMDictSettingStringsRef	DT_sStrArray	Dictionary settings strings (grid)
PMDimension	DT_POINT	Printable Area (Width,Height)
PMDisplayBoxHighlightingFlag	DT_Byte	0 – Do not use Block Highlighting 1 – Use Block Highlighting
PMDisplayColorsFlag	DT_Byte	0 – Do not display colors 1 – Display colors

Property Name	Datatype	Description
PMDisplayLeafCornersFlag	DT_Byte	0 – Do not display leaf flag on undecomposed boxes 1 – Display leaf flag on undecomposed boxes
PMDisplayShadowsFlag	DT_Byte	0 – Do not display box shadows 1 – Display box shadows
PMDisplaySquigglesFlag	DT_Byte	0 – Do not display squiggles 1 – Display squiggles

Property Name	Datatype	Description
PMDrawStyle	DT_long	Diagram Box Draw Style: 0 – Methodology Specific or standard 1 – Bitmap 2 – Shape 3 – Shape and Bitmap 4 – Defer to individual Diagram 9 – Bitmap + Name 10 – Shape + Name 11 – Shape and Bitmap + Name 17 – Bitmap + Number 18 – Shape + Number 19 – Shape and Bitmap + Number 25 – Bitmap + Name + Number 26 – Shape + Name + Number 27 – Shape and Bitmap + Name + Number 33 – Bitmap + ABC Data 34 – Shape + ABC Data 35 – Shape and Bitmap + ABC Data 41 – Bitmap + Name + ABC Data 42 – Shape + Name + ABC Data 43 – Shape and Bitmap + Name + ABC Data 49 – Bitmap + Number + ABC Data 50 – Shape + Number + ABC Data 51 – Shape and Bitmap + Number + ABC Data 57 – Bitmap + Name + Number + ABC Data 58 – Shape + Name + Number + ABC Data 59 – Shape and Bitmap + Name + Number + ABC Data
PMDurationDecimalPlaces	DT_Byte	Decimal places in duration values

Property Name	Datatype	Description
PMERSourceType	DT_Byte (Private)	0 – CA ERwin DM IDs 1 – CA ERwin MM IDs 2 – Origin IDs; new models should use this value
PMFilename	DT_LPSTR	Model File name (path)
PMFitActivityTextInBoxFlag	DT_long	Fit Names in Boxes: 0 – Do not resize or wrap 1 – Wrap text to fit box 2 – Automatically resize box to fit text
PMFrequencyDecimalPlaces	DT_Byte	Decimal places in frequency values
PMHdrFtrType0	(DT_Byte)	Left Header: 0 – None 1 – Author 2 – Author Initials 3 – C-Number 4 – CreationDate 5 – Current Date (long format) 6 – Current Date (short format) 7 – Diagram Name 8 – Diagram Number 9 – Diagram Status 10 – Model Name 11 – Model Status 12 – Page Number 13 – Project Name 14 – Revision Date
PMHdrFtrType1	(DT_Byte)	Center Header See PMHdrFtrType0 for the enumeration values
PMHdrFtrType2	(DT_Byte)	Right Header See PMHdrFtrType0 for the enumeration values

Property Name	Datatype	Description
PMHdrFtrType3	(DT_Byte)	Left Footer See PMHdrFtrType0 for the enumeration values
PMHdrFtrType4	(DT_Byte)	Center Footer See PMHdrFtrType0 for the enumeration values
PMHdrFtrType5	(DT_Byte)	Right Footer See PMHdrFtrType0 for the enumeration values
PMIsTemplate	DT_Byte	0 – Model is not a template 1 – Model is a template
PMLongDateFlag	DT_Byte	0 – Do not display dates in long format 1 – Display dates in long format
PMMergeControlFlag	DT_BOOL	SBC merge flag
PMMergeMasterModelFlag	DT_BOOL	SBC master model flag
PMMergeSubModelFlag	DT_BOOL	SBC sub-model flag
PMMethodologyCode	DT_Byte	Extended type: 0 – IDEF0 1 – DFD 2 – IDEF3 Must be the same as the Context Diagram
PMNextBoxNumber	DT_long	Next UOW number
PMNextDatastoreNumber	DT_long	Next data store number
PMNextExternalNumber	DT_long	Next external number
PMNextJunctionNumber	DT_long	Next junction number
PMNoArrowNameMergeFlag	DT_BOOL	No arrow name merge flag
PMNoAutoRenumberFlag	DT_BOOL	Use Persistent Activity numbers
PMNoDatastoreNameMergeFlag	DT_BOOL	No data store name merge flag
PMNodeTreeFontRef	DT_ref	Default node tree font reference (*PMFont*)
PMNoExternalNameMergeFlag	DT_BOOL	No external name merge flag
PMNote	DT_LPSTR	Note *DT_LPSTR*

Property Name	Datatype	Description
PMOffpageRefIdFlag	DT_long	Off Page Reference Label: 0 – C-Number 1 – Node Number 2 – Diagram Name
PMOrgDrawStyle	DT_long	Organization Chart Box Draw Style: 0 – Methodology Specific or standard 1 – Bitmap 2 – Shape 3 – Shape and Bitmap 4 – Defer to individual Diagram 9 – Bitmap + Name 10 – Shape + Name 11 – Shape and Bitmap + Name 17 – Bitmap + Number 18 – Shape + Number 19 – Shape and Bitmap + Number 25 – Bitmap + Name + Number 26 – Shape + Name + Number 27 – Shape and Bitmap + Name + Number 33 – Bitmap + ABC Data 34 – Shape + ABC Data 35 – Shape and Bitmap + ABC Data 41 – Bitmap + Name + ABC Data 42 – Shape + Name + ABC Data 43 – Shape and Bitmap + Name + ABC Data 49 – Bitmap + Number + ABC Data 50 – Shape + Number + ABC Data 51 – Shape and Bitmap + Number + ABC Data 57 – Bitmap + Name + Number + ABC Data

Property Name	Datatype	Description
		58 – Shape + Name + Number + ABC Data 59 – Shape and Bitmap + Name + Number + ABC Data
PMOriginOffset	DT_POINT	PageOffset (x,y)
PMPrefix	DT_LPSTR	Activity Number Prefix
PMProject	DT_LPSTR	Project Name
PMPurpose	DT_LPSTR	Purpose
PMReportFontRef	DT_ref	Border system text font reference (*PMFont*)
PMReportMode	DT_Byte	Cullflags: Bits 0,1-and/or, none for activity display Bits 2,3-and/or, none for arrow display Bits 4,5-and/or, none for activity reports Bits 6,7-and/or, none for arrow reports lower bit == 0-OR classes upper bit == 1-show properties with no classes Therefore, within each 2-bit field: 0x0 – Show properties that have at least one cull bit set which is also set in the mask. 0x1 – Show properties which have every cull bit set which is set in the mask. **Note:** Properties can have extra cull bits set. 0x2 – Show only those properties which have NO cull bits set in their mask. 0x3 – Show all properties (such as Ignore the cull bits).
PMRootActivityNumber	DT_long	Root activity number

Property Name	Datatype	Description
PMScope	DT_LPSTR	Scope
PMShapeIdList[0]	DT_long	IDEF0 Activity Box Shape: -1 – NotDefined 0 – BPRectangle 1 – Diamond 2 – RoundedRectangle 3 – Circle 4 – BPEllipse 5 – Pentagon 6 – Hexagon 7 – Septagon 8 – Octagon 9 – FivePointStar 10 – SixPointStar 11 – Triangle 12 – EntryPoint 13 – ExitPoint 14 – Batch 15 – Separate 16 – RightArrow 17 – LeftArrow 18 – UpArrow 19 – DownArrow 20 – CircleDecomp 21 – CircleConcentric 22 – DataStore 23 – Drum 24 – Prn

Property Name	Datatype	Description
		25 – Parall1 26 – Parall2 27 – Parall3 28 – Parall4 29 – Cross 30 – Trap1 31 – Trap2 32 – Quad1 33 – Quad2 34 – Quad3 35 – Quad4 36 – Quad5 37 – Quad6 38 – Quad7 39 – Quad8 40 – Yield 41 – RndLft 42 – RndRt 43 – RndTop 44 – RndBot
PMShapeIdList[1]	DT_long	DFD Activity Box Shape See PMShapeIdList[0] for the enumeration values
PMShapeIdList[2]	DT_long	IDEF3 Activity Box Shape See PMShapeIdList[0] for the enumeration values
PMShapeIdList[3]	DT_long	Data Store Box Shape See PMShapeIdList[0] for the enumeration values
PMShapeIdList[4]	DT_long	External Box Shape See PMShapeIdList[0] for the enumeration values

Property Name	Datatype	Description
PMShapeIdList[5]	DT_long	Referent Box Shape See PMShapeIdList[0] for the enumeration values
PMShapeIdList[6]	DT_long	Asynchronous AND Junction Box Shape See PMShapeIdList[0] for the enumeration values
PMShapeIdList[7]	DT_long	Synchronous AND Junction Box Shape See PMShapeIdList[0] for the enumeration values
PMShapeIdList[8]	DT_long	Asynchronous OR Junction Box Shape See PMShapeIdList[0] for the enumeration values
PMShapeIdList[9]	DT_long	Synchronous OR Junction Box Shape See PMShapeIdList[0] for the enumeration values
PMShapeIdList[10]	DT_long	XOR Junction Box Shape See PMShapeIdList[0] for the enumeration values
PMShowActivityCostFlag	DT_Byte	0 – Do not display ABC data in boxes 1 – Display ABC data in boxes
PMShowActivityNumberingFlag	DT_BOOL	Display Activity Numbers
PMShowActivityPrefixFlag	DT_Byte	0 – Do not display activity prefix in boxes 1 – Display activity prefix in boxes
PMShowArrowBreakFlag	DT_long	Break Arrows at Intersections: 0 – Do not break arrows 1 – Break Horizontal Arrows 2 – Break Vertical Arrows
PMShowArrowNameFlag	DT_Byte	0 – Do not display arrow names 1 – Display arrow names
PMShowDatastoreNumberingFlag	DT_BOOL	Display Data Store Numbers
PMShowDatastorePrefixFlag	DT_BOOL	Show Prefix on Data Stores

Property Name	Datatype	Description
PMShowDiagramPrefixFlag	DT_Byte	0 – Do not display prefix on diagram numbers 1 – Display prefix on diagram numbers
PMShowExternalNumberingFlag	DT_BOOL	Display External Numbers
PMShowExternalPrefixFlag	DT_BOOL	Show Prefix on Externals
PMShowICOMCodeFlag	DT_Byte	0 – Do not display ICOM codes 1 – Display ICOM codes
PMShowIDEF0KitFlag	DT_long	Header Type: 0 – No header 1 – Use IDEF0 kit 2 – Use custom header
PMShowIDEF0TitleFlag	DT_long	Footer Type: 0 – No footer 1 – Use IDEF0 title 2 – Use custom footer
PMShowTunnelFlag	DT_Byte	0 – Do not display tunnels 1 – Display tunnels
PMSortArrowsFlag	DT_Byte	0 – Do not sort arrows on box edges 1 – Sort arrows on box edges
PMSource	DT_LPSTR	Source
PMStatusString	DT_LPSTR	Status-the four standard values are: WORKING DRAFT RECOMMENDED PUBLICATION However, any string can also be used
PMSwimLaneBKColorRef	DT_ref	Default Swim Lane Background Color reference (*PMColor*)
PMSwimLaneLineColorRef	DT_ref	Default Swim Lane Line Color reference (*PMColor*)
PMSwimLaneTextColorRef	DT_ref	Default Swim Lane Text Color reference (*PMColor*)

Property Name	Datatype	Description
PMSwimLaneTextFontRef	DT_ref	Default Swim Lane text font reference (*PMFont*)
PMSystemLastRevisionDate	DT_sDate (Private)	System last revision date
PMTextBlockColorReference	DT_ref	Default Text Block Color reference (*PMColor*)
PMTextBlockFontRef	DT_ref	Default text block font reference (*PMFont*)
PMTextColorReference	DT_ref	Default Text Color reference (*PMColor*)
PMTimeFrame	DT_Byte	0 – AS-IS 1 – TO-BE
PMTimeUnit	DT_LPSTR	Time Unit
PMTxtHgtToBmpHgt	DT_double	Text height to bitmap height conversion factor
PMUserLastRevisionDate	DT_sDate	User Last Revision Date
PMViewpoint	DT_LPSTR	Viewpoint

PMActivity, PMDatastore, PMExternal, PMJunction, and PMReferent Activity Types

This section describes the valid properties for objects for all activity types, such as *PMActivity*, *PMDatastore*, *PMExternal*, *PMJunction*, and *PMReferent*.

Valid Properties for All Activity Types

The following table describes the required properties that apply to all activity types:

Property Name	Datatype	Description
PMBoxType	DT_long	Type of box: 0 – Unused Activity 1 – IDEF0 Activity 2 – DFD Activity 3 – IDEF3 Activity 4 – Data Store 5 – External 6 – Referent 7 – Asynchronous AND Junction 8 – Synchronous AND Junction 9 – Asynchronous OR Junction 10 – Synchronous OR Junction 11 – Exclusive OR Junction

The following table describes the optional properties that apply to all activity types:

Property Name	Datatype	Description
Name	DT_LPSTR	Activity name
PMAuthor	DT_LPSTR	Author
PMBackgroundColorReference	DT_ref	Activity background color index
PMConstraintsString	DT_LPSTR	Constraints
PMDefinitionString	DT_LPSTR	Definition
PMDescriptionString	DT_LPSTR	Description
PMFacts	DT_LPSTR	Facts
PMNote	DT_LPSTR	Note
PMObjectsString	DT_LPSTR	Objects
PMRRGAssocRefList	DT_sRefArray	List of RoleGroup/Role/Resource Associations

Property Name	Datatype	Description
PMSource	DT_LPSTR	Source
PMStatusString	DT_LPSTR	Status-the four standard values are: WORKING DRAFT RECOMMENDED PUBLICATION However, any string can also be used
PMTextColorReference	DT_ref	Activity text color index
PMTitleColorReference	DT_ref	Diagram title color index

PMActivity Object

The following table describes the optional properties for the *PMActivity* object:

Property Name	Datatype	Description
PMDuration	DT_double	Duration (default 0.00)
PMFrequency	DT_double	Frequency (default 1.00)
PMOverrideCostsFlag	DT_Byte	0 – Do not override costs 0x40 – Override costs
PMParentActivityRef	DT_ref	Parent activity

PMDatastore Object

The following table describes the required properties for the *PMDatastore* object:

Property Name	Datatype	Description
PMArrowRef	DT_ref	Reference to Arrow from which to inherit name
PMDatastoreNumber	DT_long	Data Store number-must be unique for each instance
PMEntityRef	DT_ref	Reference to Entity from which to inherit name

PMExternal Object

The following table describes the optional properties for the *PMExternal* object:

Property Name	Datatype	Description
PMArrowRef	DT_ref	Reference to Arrow from which to inherit name
PMEntityRef	DT_ref	Reference to Entity from which to inherit name
PMExternalNumber	DT_long	External Reference number-must be unique for each instance

PMJunction Object

The following table describes the required properties for the *PMJunction* object:

Property Name	Datatype	Description
PMJunctionBoxNumber	DT_long	Junction box number-must be unique for each instance
PMJunctionBoxTypeString	DT_LPSTR	Must be one of the following strings: Junction Box AND Asynchronous Junction Box AND Synchronous Junction Box OR Asynchronous Junction Box OR Synchronous Junction Box XOR

PMReferent Object

The following table describes the optional properties for the *PMReferent* object:

Property Name	Datatype	Description
PMArrowRef	DT_ref	Reference to Arrow from which to inherit name
PMEntityRef	DT_ref	Reference to Entity from which to inherit name

PMActivityCost Object

The following table describes the required properties for the *PMActivityCost* object:

Property Name	Datatype	Description
PMCost	DT_double	Cost
PMCostCenterRef	DT_ref	Reference to a *PMCostCenter* object
PMOverrideLeavesFlag	DT_Byte	1 – Use leaves 2 – Override leaves

PMArrow Object

The following table describes the optional properties for the *PMArrow* object:

Property Name	Datatype	Description
Name	DT_LPSTR	Arrow Name
PMAuthor	DT_LPSTR	Author
PMConstraintsString	DT_LPSTR	Constraints
PMDefinitionString	DT_LPSTR	Definition
PMDescriptionString	DT_LPSTR	Description
PMFacts	DT_LPSTR	Facts
PMMergeChangeFlag	DT_Byte	0 – SBC change flag not set 0x20 – SBC change flag set
PMNameColorReference	DT_ref	Arrow line color reference (*PMColor* object)
PMNote	DT_LPSTR	Note
PMObjectsString	DT_LPSTR	Objects

Property Name	Datatype	Description
PMStatusString	DT_LPSTR	Status-the four standard values are: WORKING DRAFT RECOMMENDED PUBLICATION However, any string can also be used
PMTextColorReference	DT_ref	Arrow name color reference (*PMColor* object)

PMArrowLabel Object

The *PMArrowLabel* object refers to an Arrow Label. The following table describes the required properties for the *PMArrowLabel* object:

Property Name	Datatype	Description
PMArrowRef	DT_ref	Reference to the *PMArrow* object of which the diagram arrow is an instance
PMLabelConnectPoint	DT_POINT	Point where a squiggle meets a label text bounding box (x,y)
PMSegmentConnectPoint	DT_POINT	Point where a squiggle meets a labeled segment (x,y)
PMTextBoundingBox	DT_RECT	The rectangle (x1,y1),(x2,y2) that determines the bounding box of the label text. x1,y1 is the upper left corner and x2,y2 is the lower right corner.

The following table describes the optional properties for the *PMArrowLabel* object:

Property Name	Datatype	Description
PMArrowColorReference	DT_ref	Arrow (or Swim Lane) line color reference (*PMColor*)
PMArrowNameColorReference	DT_ref	Arrow or Swim Lane name color reference (*PMColor*)

Property Name	Datatype	Description
PMDrawSquiggleFlag	DT_Byte	0 – Squiggle off 1 – Squiggle on
PMIsASwimLaneFlag	DT_Byte	Not used
PMSwimLaneBKColorRef	DT_ref	Not used
PMSwimLaneRoleRef	DT_ref	If the diagram arrow is a Role-based Swim Lane, then this property is a reference to a Role
PMSwimLaneStyle	DT_Byte	0x01 bit – Indicates squiggle attached to a Horizontal Segment 0x02 bit – Indicates squiggle attached to a Vertical Segment 0x04 bit – Indicates Bitmap 0x08 bit – Indicates Swim Lane 0x10 bit – Indicates Labeled
PMTextFontRef	DT_ref	Reference to the *PMFont* used for the arrow label

PMArrowSegment Object

The *PMArrowSegment* object refers to an Arrow Segment. The following table describes the required properties for the *PMArrowSegment* object:

Property Name	Datatype	Description
PMArrowDrawingStyleFlag	DT_Byte	Arrow Style: 0 – Precedence 1 – Relational 2 – Object flow 3 – Bi-directional 4 – Referent
PMLineWidth	DT_Byte	Should be in a range of 1 – 12
PMSinkConnectionRef	DT_ref	Reference to the sink object (*PMNode*, *PMBorder*, *PMBox*, *PMTunnel*)

Property Name	Datatype	Description
PMSinkTunnelStyle	DT_Byte	0 – Normal termination 1 – Tunnel 2 – Square tunnel 3 – Off page or external reference
PMSourceConnectionRef	DT_ref	Reference to the source object (*PMNode*, *PMBorder*, *PMBox*, *PMTunnel*)
PMSourceTunnelStyle	DT_Byte	0 – Normal termination 1 – Tunnel 2 – Square tunnel 3 – Off page or external reference
PMVertices	DT_sPointArray	A list of the points that define the path of the arrow from source to sink

The following table describes the optional properties for the *PMArrowSegment* object:

Property Name	Datatype	Description
PMLabeledSegment	DT_long	The zero-based index of the segment of the path to which a label is attached

PMAssociation Object

The *PMAssociation* object is an association to an entity or attribute. The following table describes the optional properties for the *PMAssociation* object:

Property Name	Datatype	Description
PMAttributeRef	DT_ref	Reference to a *PMAttribute* object
PMEntityRef	DT_ref	Reference to a *PMEntity* object
PMMergeChangeFlag	DT_Byte	0x02 – Set 0 – Not set

Note: *PMEntityRef* and *PMAttributeRef* are mutually exclusive.

PMBitmap Object

The following table describes the required properties for the *PMBitmap* object:

Property Name	Datatype	Description
Name	DT_NewCString	Bitmap Name
PMBitmapData	DT_LPNewStr	Bitmap Data

PMBorder Object

Each *PMDiagram* has four border objects. There are no object properties for *PMBorder*.

PMBox Object

The following table describes the required properties for the *PMBox* object:

Property Name	Datatype	Description
PMBoxCoordinates	DT_RECT	The rectangle (x1,y1),(x2,y2) that determines the size of the box. x1,y1 is the upper left corner and x2,y2 is the lower right corner.
PMDrawStyle	DT_Byte	0 – Standard 1 – Bitmap 2 – Shape 3 – Shape and bitmap Additional bits: 0x08 – Show name 0x10 – Show number 0x20 – Show ABC data
PMPersistentBoxNumber	DT_long	Used in IDEF0 and DFD diagrams
PMUowIdNumber	DT_long	Used in IDEF3 and Swim Lane diagrams

The following table describes the optional properties for the *PMBox* object:

Property Name	Datatype	Description
PMActivityRef	DT_ref	A reference to the activity, data store, external reference, referent, or junction represented by the box
PMBackgroundColorReference	DT_ref	Background color reference (*PMColor*)
PMBitmapJustification	DT_long	Justification of bitmap in box: 0 – Left justified text 1 – Center justified text 2 – Right justified text
PMBitmapRef	DT_ref	Reference to bitmap associated with this box
PMObjectFontReference	DT_ref	Name text font reference (*PMFont*)

Property Name	Datatype	Description
PMShapeId	DT_long	Index of shape associated with box: -1 – NotDefined 0 – BPRectangle 1 – Diamond 2 – RoundedRectangle 3 – Circle 4 – BPEllipse 5 – Pentagon 6 – Hexagon 7 – Septagon 8 – Octagon 9 – FivePointStar 10 – SixPointStar 11 – Triangle 12 – EntryPoint 13 – ExitPoint 14 – Batch 15 – Separate 16 – RightArrow 17 – LeftArrow 18 – UpArrow 19 – DownArrow 20 – CircleDecomp 21 – CircleConcentric 22 – DataStore 23 – Drum 24 – Prn

Property Name	Datatype	Description
		25 – Parall1 26 – Parall2 27 – Parall3 28 – Parall4 29 – Cross 30 – Trap1 31 – Trap2 32 – Quad1 33 – Quad2 34 – Quad3 35 – Quad4 36 – Quad5 37 – Quad6 38 – Quad7 39 – Quad8 40 – Yield 41 – RndLft 42 – RndRt 43 – RndTop 44 – RndBot
PMTextColorReference	DT_ref	Text color reference (*PMColor*)
PMTitleColorReference	DT_ref	Decomposition diagram title color reference (*PMColor*)

PMColor Object

The following table describes the required properties for the *PMColor* object:

Property Name	Datatype	Description
PMBlueIntensity	DT_Byte	Blue Intensity 0-255
PMGreenIntensity	DT_Byte	Green Intensity 0-255

Property Name	Datatype	Description
PMRedIntensity	DT_Byte	Red Intensity 0-255
PMSequenceNumber	DT_long	Palette index

PMCostCenter Object

The following table describes the required properties for the *PMCostCenter* object:

Property Name	Datatype	Description
Name	DT_LPSTR	Cost Center name

The following table describes the optional properties for the *PMCostCenter* object:

Property Name	Datatype	Description
PMDefinitionString	DT_LPSTR	Definition

PMCRUD Object

The following table describes the required properties for the *PMCRUD* object:

Property Name	Datatype	Description
PMActivityRef	DT_ref	Reference to a *PMActivity* object
PMArrowType	DT_char	'I', 'C', 'O', 'M' (Input,Control,Output,Mechanism)

The following table describes the optional properties for the *PMCRUD* object:

Property Name	Datatype	Description
PMCreateFlag	DT_Byte	0 – Not set 1 – Set
PMDeleteFlag	DT_Byte	0 – Not set 8 – Set

Property Name	Datatype	Description
PMMergeChangeFlag	DT_Byte	0 – Not set 128 – Set
PMReadFlag	DT_Byte	0 – Not set 2 – Set
PMUpdateFlag	DT_Byte	0 – Not set 4 – Set

PMDiagram and PMNodeTree Diagram Types (Includes Organization Charts)

This section describes the valid common properties for objects such as *PMDiagram* and *PMNodeTree*.

The following table describes the properties for all diagram types:

Property Name	Datatype	Description
Name	DT_LPSTR	Diagram Name
PMActiveAreaDimension	DT_POINT	Size of active area of diagram
PMArcRadius	DT_long	Arc radius for arcs on arrows
PMBitmapJustification	DT_long	Bitmap justification in box: 0 – Left justified text 1 – Center justified text 2 – Right justified text
PMBorderSystemDefaultFontRef	DT_ref	Kit and title text font reference (*PMFont*)
PMBorderUserFontRef	DT_ref	Title and node number font reference (*PMFont*)
PMCNumber	DT_LPSTR	C-Number
PMCreatedDate	DT_sDate	Diagram creation date
PMCurrentDiagram	DT_Byte	0 – Not the active diagram 0x10 – Diagram is active diagram (there can be only one)
PMDecompositionNumber	DT_long	Decomposition number (IDEF3)
PMDecompositionOfRef	DT_ref	Reference to a *PMBox* object of a Parent Activity

Property Name	Datatype	Description
PMDefaultActivityFontRef	DT_ref	New activity font reference (*PMFont*)
PMDefaultArrowFontRef	DT_ref	New arrow label font reference (*PMFont*)
PMDiagramNumber	DT_LPSTR	Diagram Number
PMDimension	DT_POINT	Sheet size
PMDisplayOffset	DT_POINT	Offset from the upper left corner of the window to the origin of the diagram. (0,-96) is a good default.
PMDrawStyle	DT_Byte	A flag byte to indicate the representation to use, as follows: 0 – Standard representation 1 – Bitmap 2 – Shape 3 – Shape and bitmap (Node Tree only) 4 – Defer to individual box Additional bits: 0x08 – Show name 0x10 – Show number 0x20 – Show ABC data

Property Name	Datatype	Description
PMHdrFtrType0	DT_long	Byte flag indicating the choices for the upper left header field. The values correspond to the indexes in the combo boxes. 0 – NoField, 1 – ModelName 2 – ProjectName 3 – DiagramName 4 – DiagramNumber 5 – CNumber 6 – PageNumber 7 – ShortDateField 8 – ShortDateTime 9 – LongDateField 10 – AuthorField 11 – AuthorInitials 12 – CreationDate 13 – RevisionDate 14 – DiagramStatus 15 – ModelStatus
PMHdrFtrType1	DT_long	Byte flag indicating the choices for the upper center header field. The values correspond to the indexes in the combo boxes. See PMHdrFtrType0 for the enumeration values
PMHdrFtrType2	DT_long	Byte flag indicating the choices for the upper right header field. The values correspond to the indexes in the combo boxes. See PMHdrFtrType0 for the enumeration values

Property Name	Datatype	Description
PMHdrFtrType3	DT_long	Byte flag indicating the choices for the lower left footer field. The values correspond to the indexes in the combo boxes. See PMHdrFtrType0 for the enumeration values
PMHdrFtrType4	DT_long	Byte flag indicating the choices for the lower center footer field. The values correspond to the indexes in the combo boxes. See PMHdrFtrType0 for the enumeration values
PMHdrFtrType5	DT_long	Byte flag indicating the choices for the lower right footer field. The values correspond to the indexes in the combo boxes. See PMHdrFtrType0 for the enumeration values
PMMethodologyCode	DT_Byte	Extended type: 0 – IDEF0 1 – DFD 2 – IDEF3 3 – Swim Lane 4 – Organization Chart
PMOriginOffset	DT_POINT	Offset to (0,0) from upper left corner of sheet
PMPageNumberString	DT_LPSTR	Page Number
PMParentDiagramRef	DT_ref	Reference to parent diagram (*PMDiagram*)
PMRepresentedActivityRef	DT_ref	Reference to parent activity (*PMActivity*)
PMShowIDEF0KitFlag	DT_Byte	0 – No header 1 – Kit 2 – Alternative header
PMShowIDEF0TitleFlag	DT_Byte	0 – No footer 1 – Title 2 – Alternative footer

Property Name	Datatype	Description
PMStatusString	DT_LPSTR	Status-the four standard values are: WORKING DRAFT RECOMMENDED PUBLICATION However, any string can also be used
PMSwimLaneRoleRef	DT_ref	Reference to a *PMRoleType* used for swim lanes
PMSwimLaneRoleType	DT_Byte	0 – Not a Swim Lane diagram (this property will not be present if it is not a Swim Lane diagram) 1 – UDP-based 2 – Role-based
PMSwimLaneUDPRef	DT_ref	Reference to a single-selection *PMUDPDefinition* used for swim lanes
PMSystemLastRevisionDate	DT_sDate	System Last Revision Date
PMTextBlockFontRef	DT_ref	New text block font reference (*PMFont*)
PMTextFontRef	DT_ref	Parent diagram text font reference (*PMFont*)
PMTextString	DT_LPSTR	Diagram Text
PMTitleFontRef	DT_ref	Parent diagram title font reference (*PMFont*)
PMTxtHgtToBmpHgt	DT_double	Text height to bitmap height conversion factor
PMUsedAtString	DT_LPSTR	Used At String
PMUserLastRevisionDate	DT_sDate	User Last Revision Date
PMVersion	DT_long	CA ERwin MM revision number when the diagram graphics were last updated
PMZoomScale	DT_long	Scale factor (shown as a percentage) the last time diagram was displayed (75 percent is a good arbitrary default)

PMDiagram

The following table describes the required properties for the *PMDiagram* object, and is valid for IDEF0, DFD, IDEF3, FEO, Scenario, and Swim Lane:

Property Name	Datatype	Description
PMFEOFlag	DT_long	0 – Not a FEO 1 – Is a FEO (for exposition only) diagram

The following table describes the optional properties for the *PMDiagram* object, and is valid for IDEF0, DFD, IDEF3, FEO, Scenario, and Swim Lane:

Property Name	Datatype	Description
PMBottomRef	DT_ref	Reference to a bottom border object (*PMBorder*)
PMLeftRef	DT_ref	Reference to a left border object (*PMBorder*)
PMRightRef	DT_ref	Reference to a right border object (*PMBorder*)
PMScenario	DT_LPSTR	Scenario name
PMTopRef	DT_ref	Reference to a top border object (*PMBorder*)

PMNodeTree

The following table describes the optional properties for the *PMNodeTree* object, and is valid for Node Tree and Org Chart:

Property Name	Datatype	Description
PMBitmapHeight	DT_long	Not used
PMBoxDisplayFlags	DT_Byte	OR of the following values: 0x02 – Show role name text on Organization Chart box 0x04 – Show resource name text on Organization Chart box 0x08 – Show role group name text on Organization Chart box

Property Name	Datatype	Description
PMBoxWidthPrefFlag	DT_Byte	0 – Fit text to each box 1 – One size per row 2 – All one size
PMBulletLastFlag	DT_Byte	0 – Do not bullet last level 1 – Bullet last level
PMDefaultBoxWidth	DT_long	Not used
PMDrawBorderFlag	DT_Byte	0 – Do not draw box borders 1 – Draw box borders
PMDrawBoxFlag	DT_Byte	0 – Do not draw boxes 1 – Draw boxes
PMNumberInBoxFlag	DT_Byte	0 – Do not draw numbers 1 – Draw numbers
PMNumLevels	DT_long	Number of levels to show in the tree
PMOrthogonalFlag	DT_Byte	0 – Diagonal lines 1 – Orthogonal lines

PMERwinAttribute Object

The following table describes the required properties for the *PMERwinAttribute* object:

Property Name	Datatype	Description
Name	DT_LPSTR	Name of Attribute
PMERAttributeId	DT_LPPMID	20-byte ID of a CA ERwin DM Attribute (if owned by CA ERwin DM)

The following table describes the optional properties for the *PMERwinAttribute* object:

Property Name	Datatype	Description
PMBPwinGeneratedFlag	DT_long	0 – Owned by CA ERwin DM 1 – Not yet exported to CA ERwin DM
PMBPwinOnlyFlag	DT_long	0 – Owned by CA ERwin DM 1 – Not to be exported to CA ERwin DM
PMDefinitionString	DT_LPSTR	Attribute definition
PMERDataType	DT_LPSTR	CA ERwin DM Data Type (logical)
PMParentname	DT_LPSTR	Base Name if Name is Role Name

PMERwinEntity Object

The following table describes the required properties for the *PMERwinEntity* object:

Property Name	Datatype	Description
Name	DT_LPSTR	Name of Entity
PMEREntityId	DT_LPPMID	20-byte ID of a CA ERwin DM Entity (if owned by CA ERwin DM)

The following table describes the optional properties for the *PMERwinEntity* object:

Property Name	Datatype	Description
PMBPwinGeneratedFlag	DT_long	0 – Owned by CA ERwin DM 1 – Not yet exported to CA ERwin DM
PMBPWinOnlyFlag	DT_long	0 – Owned by CA ERwin DM 1 – Not to be exported to CA ERwin DM
PMDefinitionString	DT_LPSTR	Entity Definition
PMERModelRef	DT_ref	Reference to a *PMERModel* that owns this entity

PMERwinModel Object

The following table describes the required properties for the *PMERwinModel* object:

Property Name	Datatype	Description
Name	DT_LPSTR	Name of a CA ERwin DM Model
PMERModelMMId	DT_LPPMID	20-byte ID of a CA ERwin DM Model

PMFont Object

The following table describes the required properties for the *PMFont* object:

Property Name	Datatype	Description
PMLogfont	DT_LPLOGFONT	Pointer to LOGFONT structure

The following table describes the optional properties for the *PMFont* object:

Property Name	Datatype	Description
PMFontSize	DT_long	Height in 1/10th points

PMIRUN Object

The following table describes the required properties for the *PMIRUN* object:

Property Name	Datatype	Description
PMActivityRef	DT_ref	Reference to a *PMActivity* object
PMArrowType	DT_char	'I', 'C', 'O', 'M' (Input,Control,Output,Mechanism)

The following table describes the optional properties for the *PMIRUN* object:

Property Name	Datatype	Description
PMInsertFlag	DT_Byte	0 – Not set 1 – Set
PMMergeChangeFlag	DT_Byte	0 – Not set 128 – Set
PMNullFlag	DT_Byte	0 – Not set 8 – Set
PMReadFlag	DT_Byte	0 – Not set 2 – Set
PMUpdateFlag	DT_Byte	0 – Not set 4 – Set

PMNode Object

PMNode describes points where arrows branch or join. The following table describes the required properties for the *PMNode* object:

Property Name	Datatype	Description
PMPointLocation	DT_Point	Coordinate of node (x,y)

PMOrgChartBox Object

The following table describes the required properties for the *PMOrgChartBox* object:

Property Name	Datatype	Description
PMBoxCoordinates	DT_RECT	The rectangle (x1,y1),(x2,y2) that determines the size of the box. x1,y1 is the upper left corner and x2,y2 is the lower right corner.
PMOrgBoxHasParent	DT_long	0 – Is not a parent of other boxes 1 – Is a parent of other boxes
PMOrgBoxId	DT_long	Box position (breadth first order)

Property Name	Datatype	Description
PMOrgBoxParentId	DT_long	Index of Parent in Parent Row
PMRRGAssocRef	DT_ref	Activity (*PMActivity*) or Role (*PMRole*) reference

The following table describes the optional properties for the *PMOrgChartBox* object:

Property Name	Datatype	Description
PMOrgBoxBKColorReference	DT_ref	Background color reference
PMShapeColorReference	DT_ref	Shape color reference (*PMColor*)
PMTextColorReference	DT_ref	Text color reference (*PMColor*)

PMOrgChartRow Object

The following table describes the required properties for the *PMOrgChartRow* object:

Property Name	Datatype	Description
PMFirstBoxInRow	DT_long	ID of first box in a row
PMMaxRowBoxCount	DT_long	Allocated count of boxes in a row
PMOrgRowId	DT_long	Row number (from first row = 0)
PMRowBoxCount	DT_long	Count of boxes in a row

PMPalette Object

The following table describes the required properties for the *PMPalette* object:

Property Name	Datatype	Description
PMVersion	DT_long	Palette Version (must always be set to 0x300)

PMResource Object

The following table describes the required properties for the *PMResource* object:

Property Name	Datatype	Description
NameCopy	DT_NewCString	Resource Name

The following table describes the optional properties for the *PMResource* object:

Property Name	Datatype	Description
PMDescriptionStringCopy	DT_NewCString	Description

PMRole Object

The following table describes the required properties for the *PMRole* object:

Property Name	Datatype	Description
Name	DT_NewCString	Role Name
PMBitmapRef	DT_ref	Reference to Associated Bitmap
PMImportance	DT_long	Importance: 0 – Low 1 – Medium 2 – High
PMShapeId	DT_long	Associated Shape: See the *PMShapeId* property of the *PMRoleGroup* object for the enumeration values

The following table describes the optional properties for the *PMRole* object:

Property Name	Datatype	Description
PMDescriptionStringCopy	DT_NewCString	Description

PMRoleGroup Object

The following table describes the required properties for the *PMRoleGroup* object:

Property Name	Datatype	Description
Name	DT_NewCString	Role Group Name
PMBitmapRef	DT_ref	Reference to Associated Bitmap
PMImportance	DT_long	Importance: 0 – Low 1 – Medium 2 – High

Property Name	Datatype	Description
PMShapeId	DT_long	Associated Shape: -1 – NotDefined 0 – BPRectangle 1 – Diamond 2 – RoundedRectangle 3 – Circle 4 – BPEllipse 5 – Pentagon 6 – Hexagon 7 – Septagon 8 – Octagon 9 – FivePointStar 10 – SixPointStar 11 – Triangle 12 – EntryPoint 13 – ExitPoint 14 – Batch 15 – Separate 16 – RightArrow 17 – LeftArrow 18 – UpArrow 19 – DownArrow 20 – CircleDecomp 21 – CircleConcentric 22 – DataStore 23 – Drum 24 – Prn

Property Name	Datatype	Description
		25 – Parall1
		26 – Parall2
		27 – Parall3
		28 – Parall4
		29 – Cross
		30 – Trap1
		31 – Trap2
		32 – Quad1
		33 – Quad2
		34 – Quad3
		35 – Quad4
		36 – Quad5
		37 – Quad6
		38 – Quad7
		39 – Quad8
		40 – Yield
		41 – RndLft
		42 – RndRt
		43 – RndTop
		44 – RndBot

The following table describes the optional properties for the *PMRoleGroup* object:

Property Name	Datatype	Description
PMDescriptionStringCopy	DT_NewCString	Description

PMRRGAssociation Object

The following table describes the required properties for the *PMRRGAssociation* object:

Property Name	Datatype	Description
PMRoleTypeRef	DT_ref	Reference to a *PMRoleType* object
PMRoleRef	DT_ref	Reference to a *PMRole* object

The following table describes the optional properties for the *PMRRGAssociation* object:

Property Name	Datatype	Description
PMResourceRef	DT_ref	Reference to a *PMResource* object

PMRuler Object

The PMRuler object is used only in Swim Lanes. The following table describes the required properties for the *PMRuler* object:

Property Name	Datatype	Description
PMRulerUnitsUDPRef	DT_ref	Reference to *PMUDPDefinition* for the UDP upon which the ruler units are based

The following table describes the optional properties for the *PMRuler* object:

Property Name	Datatype	Description
PMRulerBKColorRef	DT_ref	Background color reference (*PMColor*)
PMRulerDisplayFlags	DT_Byte	0x01 – Display ruler at the top of the diagram 0x02 – Display ruler at the bottom of the diagram
PMRulerLabel	DT_LPSTR	Label DT_LPSTR
PMRulerLabelColorRef	DT_ref	Ruler label color reference (*PMColor*)

Property Name	Datatype	Description
PMRulerLeftValueDate	DT_sDate	Leftmost value of a scale if it is a date
PMRulerLeftValueDbl	DT_double	Leftmost value of a scale if it is double
PMRulerLeftValueLong	DT_long	Leftmost value of a scale if it is long
PMRulerMajorDivColorRef	DT_ref	Major division color reference (*PMColor*)
PMRulerMajorIncDbl	DT_double	Distance between major increments if it is double
PMRulerMajorIncLong	DT_long	Distance between major increments if it is long or date
PMRulerMinorDivColorRef	DT_ref	Minor division color reference (*PMColor*)
PMRulerMinorIncDbl	DT_double	Distance between minor increments if it is double
PMRulerMinorIncLong	DT_long	Distance between minor increments if it is long or date
PMRulerRightValueDate	DT_sDate	Rightmost value of a scale if it is a date
PMRulerRightValueDbl	DT_double	Rightmost value of a scale if it is double
PMRulerRightValueLong	DT_long	Rightmost value of a scale if it is long
PMRulerStyle	DT_long	Style of ruler (not used, set to 0)
PMRulerUnitsColorRef	DT_ref	Ruler units color reference (*PMColor*)
PMRulerUnitsLabelFontRef	DT_ref	Ruler units label color reference (*PMFont*)
PMRulerUnitsTextFontRef	DT_ref	Ruler units color reference (*PMFont*)

PMTextBlock Object

The following table describes the required properties for the *PMTextBlock* object:

Property Name	Datatype	Description
PMTextBoundingBox	DT_RECT	The rectangle (x1,y1),(x2,y2) that determines the bounding box of the text block. x1,y1 is the upper left corner and x2,y2 is the lower right corner.
PMTextString	DT_LPSTR	The actual text string

The following table describes the optional properties for the *PMTextBlock* object:

Property Name	Datatype	Description
PMIsPurposeFlag	DT_long	0 – Use the *PMTextString* 2 – Use Model Purpose instead of the *PMTextString*
PMIsViewpointFlag	DT_long	0 – Use the *PMTextString* 1 – Use Model Viewpoint instead of the *PMTextString*
PMTextBlockColorReference	DT_ref	Text color reference (*PMColor*)
PMTextBlockFontReference	DT_ref	Text font reference (*PMFont*)

Note: *PMIsPurposeFlag* and *PMIsViewpointFlag* are mutually exclusive.

PMTunnel Object

The *PMTunnel* object is for Off Page References and Externals and not for Tunnels. The following table describes the required properties for the *PMTunnel* object; use one or the other property depending on type:

Property Name	Datatype	Description
PMExternalRef	DT_ref	Reference to an External Reference
PMOffpageRef	DT_ref	Reference to the parent activity of the referenced diagram

PMUDPCategory Object

The following table describes the required properties for the *PMUDPCategory* object:

Property Name	Datatype	Description
Name	DT_LPSTR	Category Name
PMSequenceNumber	DT_long	0-based index-must be unique

The following table describes the optional properties for the *PMUDPCategory* object:

Property Name	Datatype	Description
PMDispInActivityDispFlag	DT_BOOL	TRUE to display property in Activity Property Page
PMDispInActivityRptFlag	DT_BOOL	TRUE to display property in Activity Reports
PMDispInArrowDispFlag	DT_BOOL	TRUE to display property in Arrow Property Page
PMDispInArrowRptFlag	DT_BOOL	TRUE to display property in Arrow Reports

PMUDPDefinition Object

The following table describes the required properties for the *PMUDPDefinition* object:

Property Name	Datatype	Description
Name	DT_LPSTR	Name of the UDP
PMValueType	DT_long	Type of UDP: 0 – Unused 1 – Text 2 – Paragraph 3 – Integer 4 – CommandLine 5 – Char 6 – Date 7 – Real 8 – TextSet 9 – IntegerSet 10 – CommandSet 11 – DateSet 12 – RealSet 13 – CharSet

The following table describes the optional properties for the *PMUDPDefinition* object:

Property Name	Datatype	Description
PMCategories	DT_sRefArray	UDP Categories
PMDateStyleFlag	DT_Byte	0 – Do not display dates in long format 1 – Display dates in long format
PMPrecision	DT_Byte	Decimal places in floating point values
PMSequenceNumber	DT_long	0-based index-must be unique for each instance

Property Name	Datatype	Description
PMUDPDescription	DT_LPSTR	Descriptive text
PMUDPMultiSelectFlag	DT_Byte	0 – Single select list 0x80 – Multi select list

PMUDPInstance Object

The following table describes the required properties for the *PMUDPInstance* object:

Property Name	Datatype	Description
PMDefinedByRef	DT_ref	Reference to the *PMUDPDefinition*
PMValueOfRef	DT_ref	Reference to the *PMUDPValue* (only for single-select list UDP)
PMValueOfRefSet	DT_sRefArray	Reference to the selection array (only for multi-select UDP)

PMUDPValue Object

The following table describes the optional properties for the *PMUDPValue* object:

Property Name	Datatype	Description
PMDateUDPValue	DT_sDate	Date type UDP value
PMFloatUDPValue	DT_double	Real type UDP value
PMIntUDPValue	DT_long	Integer type UDP
PMSequenceUDPValue	DT_short	Single-select UDP selection
PMStringUDPValue	DT_LPSTR	Text, Paragraph, Command, or Character type UDP value

Note: Only one of these properties should belong to an instance, based on the type of the *UDPDefinition* or the *UDPInstance* that owns the instance.

Identifiers

The following tables describe the values used in identifying object types and their properties in the CA ERwin PM API interface functions.

The identifiers for object and property types in CA ERwin PM are all longs and not GUIDs. The API uses GUIDs for all identifiers. To handle this situation, this scheme is used: the first set of data in the GUID is the hexadecimal equivalent of the identifier for the object or property type; the rest of the fields in the GUID are zeros. For example, if the GUID for an object or a property type is:

{000003E8-0000-0000-0000-000000000000}+00000000

The type identifier is 0x3E8 or 1000, which is a *PMDiagram*.

Object Types

The following table lists the CA ERwin PM API object types:

Class Name	Class ID	Long
PMActivity	{000003ED-0000-0000-0000-000000000000}+00000000	1005
PMActivityCost	{000003DD-0000-0000-0000-000000000000}+00000000	989
PMArrow	{000003DE-0000-0000-0000-000000000000}+00000000	990
PMArrowLabel	{000003E0-0000-0000-0000-000000000000}+00000000	992
PMArrowSegment	{000003DF-0000-0000-0000-000000000000}+00000000	991
PMAssociation	{000003E1-0000-0000-0000-000000000000}+00000000	993
PMBitmap	{000004BE-0000-0000-0000-000000000000}+00000000	1214
PMBorder	{000003E2-0000-0000-0000-000000000000}+00000000	994
PMBox	{000003E3-0000-0000-0000-000000000000}+00000000	995
PMColor	{000003E4-0000-0000-0000-000000000000}+00000000	996
PMCostCenter	{000003E5-0000-0000-0000-000000000000}+00000000	997
PMCRUD	{000003E6-0000-0000-0000-000000000000}+00000000	998
PMDataStore	{000003E7-0000-0000-0000-000000000000}+00000000	999
PMDiagram	{000003E8-0000-0000-0000-000000000000}+00000000	1000
PMERwinAttribute	{000003E9-0000-0000-0000-000000000000}+00000000	1001
PMERwinEntity	{000003EA-0000-0000-0000-000000000000}+00000000	1002
PMERwinModel	{00000515-0000-0000-0000-000000000000}+00000000	1301

Class Name	Class ID	Long
PMExternal	{000003EB-0000-0000-0000-000000000000}+00000000	1003
PMFont	{000003EC-0000-0000-0000-000000000000}+00000000	1004
PMIRUN	{000003EE-0000-0000-0000-000000000000}+00000000	1006
PMJunction	{0000040A-0000-0000-0000-000000000000}+00000000	1034
PMModel	{000003EF-0000-0000-0000-000000000000}+00000000	1007
PMNode	{000003F0-0000-0000-0000-000000000000}+00000000	1008
PMNodeTree	{000003F1-0000-0000-0000-000000000000}+00000000	1009
PMOrgChartBox	{000004BC-0000-0000-0000-000000000000}+00000000	1212
PMOrgChartRow	{000004BB-0000-0000-0000-000000000000}+00000000	1211
PMPalette	{00000426-0000-0000-0000-000000000000}+00000000	1062
PMReferent	{000003F2-0000-0000-0000-000000000000}+00000000	1010
PMResource	{000004C1-0000-0000-0000-000000000000}+00000000	1217
PMRole	{000004C0-0000-0000-0000-000000000000}+00000000	1216
PMRoleGroup	{000004BF-0000-0000-0000-000000000000}+00000000	1215
PMRRGAssociation	{000004C2-0000-0000-0000-000000000000}+00000000	1218
PMRuler	{000004BD-0000-0000-0000-000000000000}+00000000	1213
PMTextBlock	{000003F3-0000-0000-0000-000000000000}+00000000	1011
PMTunnel	{000003F4-0000-0000-0000-000000000000}+00000000	1012
PMUDPCategory	{000003F5-0000-0000-0000-000000000000}+00000000	1013
PMUDPDefinition	{000003F6-0000-0000-0000-000000000000}+00000000	1014
PMUDPInstance	{000003F7-0000-0000-0000-000000000000}+00000000	1015
PMUDPValue	{000003FA-0000-0000-0000-000000000000}+00000000	1018

Properties

Each CA ERwin PM API object type has its own section, and a table in each section describes the property class name, class ID, and long value of the object type.

PMActivity

The following table describes the property class name, class ID, and long value for the *PMActivity* object type:

Property Class Name	Class ID	Long
PMAuthor	{0000031C-0000-0000-0000-000000000000}+00000000	796
PMBackgroundColorReference	{00000320-0000-0000-0000-000000000000}+00000000	800
PMBoxType	{000004E8-0000-0000-0000-000000000000}+00000000	1256
PMConstraintsString	{00000333-0000-0000-0000-000000000000}+00000000	819
PMDefinitionString	{00000346-0000-0000-0000-000000000000}+00000000	838
PMDescriptionString	{00000348-0000-0000-0000-000000000000}+00000000	840
PMDuration	{00000356-0000-0000-0000-000000000000}+00000000	854
PMFacts	{00000361-0000-0000-0000-000000000000}+00000000	865
PMFrequency	{00000364-0000-0000-0000-000000000000}+00000000	868
PMNote	{0000037E-0000-0000-0000-000000000000}+00000000	894
PMObjectsString	{00000384-0000-0000-0000-000000000000}+00000000	900
PMOverrideCostsFlag	{00000389-0000-0000-0000-000000000000}+00000000	905
PMParentActivityRef	{0000038C-0000-0000-0000-000000000000}+00000000	908
PMRRGAssocRefList	{000004E0-0000-0000-0000-000000000000}+00000000	1248
PMSource	{000003AF-0000-0000-0000-000000000000}+00000000	943
PMStatusString	{000003B3-0000-0000-0000-000000000000}+00000000	947
PMTextColorReference	{000003BA-0000-0000-0000-000000000000}+00000000	954
PMTitleColorReference	{000003BF-0000-0000-0000-000000000000}+00000000	959
Name	{00000093-0000-0000-0000-000000000000}+00000000	147

PMActivityCost

The following table describes the property class name, class ID, and long value for the *PMActivityCost* object type:

Property Class Name	Class ID	Long
PMCost	{00000336-0000-0000-0000-000000000000}+00000000	822
PMCostCenterRef	{00000337-0000-0000-0000-000000000000}+00000000	823
PMOverrideLeavesFlag	{0000038A-0000-0000-0000-000000000000}+00000000	906

PMArrow

The following table describes the property class name, class ID, and long value for the *PMArrow* object type:

Property Class Name	Class ID	Long
PMAuthor	{0000031C-0000-0000-0000-000000000000}+00000000	796
PMConstraintsString	{00000333-0000-0000-0000-000000000000}+00000000	819
PMDefinitionString	{00000346-0000-0000-0000-000000000000}+00000000	838
PMDescriptionString	{00000348-0000-0000-0000-000000000000}+00000000	840
PMFacts	{00000361-0000-0000-0000-000000000000}+00000000	865
PMMergeChangeFlag	{000004B8-0000-0000-0000-000000000000}+00000000	1208
PMNameColorReference	{00000373-0000-0000-0000-000000000000}+00000000	883
PMNote	{0000037E-0000-0000-0000-000000000000}+00000000	894
PMObjectsString	{00000384-0000-0000-0000-000000000000}+00000000	900
PMStatusString	{000003B3-0000-0000-0000-000000000000}+00000000	947
PMTextColorReference	{000003BA-0000-0000-0000-000000000000}+00000000	954
Name	{00000093-0000-0000-0000-000000000000}+00000000	147

PMArrowLabel

The following table describes the property class name, class ID, and long value for the *PMArrowLabel* object type:

Property Class Name	Class ID	Long
PMArrowColorReference	{00000316-0000-0000-0000-000000000000}+00000000	790
PMArrowNameColorReference	{00000317-0000-0000-0000-000000000000}+00000000	791
PMArrowRef	{00000318-0000-0000-0000-000000000000}+00000000	792
PMDrawSquiggleFlag	{00000355-0000-0000-0000-000000000000}+00000000	853
PMIsASwimLaneFlag	{000004C4-0000-0000-0000-000000000000}+00000000	1220
PMLabelConnectPoint	{0000036B-0000-0000-0000-000000000000}+00000000	875
PMSegmentConnectPoint	{0000039D-0000-0000-0000-000000000000}+00000000	925
PMSwimLaneBKColorRef	{000004C5-0000-0000-0000-000000000000}+00000000	1221
PMSwimLaneRoleRef	{000004C7-0000-0000-0000-000000000000}+00000000	1223
PMSwimLaneStyle	{000004E9-0000-0000-0000-000000000000}+00000000	1257
PMSwimLaneUDPRef	{000004C6-0000-0000-0000-000000000000}+00000000	1222
PMTextBoundingBox	{000003B9-0000-0000-0000-000000000000}+00000000	953
PMTextFontRef	{000003BB-0000-0000-0000-000000000000}+00000000	955

PMArrowSegment

The following table describes the property class name, class ID, and long value for the *PMArrowSegment* object type:

Property Class Name	Class ID	Long
PMArrowDrawingStyleFlag	{00000420-0000-0000-0000-000000000000}+00000000	1056
PMLabeledSegment	{0000036C-0000-0000-0000-000000000000}+00000000	876
PMLineWidth	{000004B3-0000-0000-0000-000000000000}+00000000	1203
PMSinkConnectionRef	{000003AC-0000-0000-0000-000000000000}+00000000	940
PMSinkTunnelStyleFlag	{000003AE-0000-0000-0000-000000000000}+00000000	942
PMSourceConnectionRef	{000003B0-0000-0000-0000-000000000000}+00000000	944
PMSourceTunnelStyleFlag	{000003B2-0000-0000-0000-000000000000}+00000000	946
PMVertices	{000003C9-0000-0000-0000-000000000000}+00000000	969

PMAssociation

The following table describes the property class name, class ID, and long value for the *PMAssociation* object type:

Property Class Name	Class ID	Long
PMAttributeRef	{0000031B-0000-0000-0000-000000000000}+00000000	795
PMEntityRef	{0000035D-0000-0000-0000-000000000000}+00000000	861
PMMergeChangeFlag	{000004B8-0000-0000-0000-000000000000}+00000000	1208

PMBitmap

The following table describes the property class name, class ID, and long value for the *PMBitmap* object type:

Property Class Name	Class ID	Long
PMBitmapData	{000004FC-0000-0000-0000-000000000000}+00000000	1276
NameCopy	{00000094-0000-0000-0000-000000000000}+00000000	148

PMBox

The following table describes the property class name, class ID, and long value for the *PMBox* object type:

Property Class Name	Class ID	Long
PMActivityRef	{00000314-0000-0000-0000-000000000000}+00000000	788
PMBackgroundColorReference	{00000320-0000-0000-0000-000000000000}+00000000	800
PMBitmapJustification	{000004CB-0000-0000-0000-000000000000}+00000000	1227
PMBitmapRef	{000004C8-0000-0000-0000-000000000000}+00000000	1224
PMBoxCoordinates	{0000044E-0000-0000-0000-000000000000}+00000000	1102
PMDrawStyle	{000004CA-0000-0000-0000-000000000000}+00000000	1226
PMObjectFontReference	{00000383-0000-0000-0000-000000000000}+00000000	899
PMPersistentBoxNumber	{0000038E-0000-0000-0000-000000000000}+00000000	910
PMShapeId	{000004C9-0000-0000-0000-000000000000}+00000000	1225
PMTextColorReference	{000003BA-0000-0000-0000-000000000000}+00000000	954

Property Class Name	Class ID	Long
PMTitleColorReference	{000003BF-0000-0000-0000-000000000000}+00000000	959
PMUowIdNumber	{000003C3-0000-0000-0000-000000000000}+00000000	963

PMColor

The following table describes the property class name, class ID, and long value for the *PMColor* object type:

Property Class Name	Class ID	Long
PMBlueIntensity	{00000423-0000-0000-0000-000000000000}+00000000	1059
PMGreenIntensity	{00000422-0000-0000-0000-000000000000}+00000000	1058
PMRedIntensity	{00000421-0000-0000-0000-000000000000}+00000000	1057
PMSequenceNumber	{0000039E-0000-0000-0000-000000000000}+00000000	926

PMCostCenter

The following table describes the property class name, class ID, and long value for the *PMCostCenter* object type:

Property Class Name	Class ID	Long
PMDefinitionString	{00000346-0000-0000-0000-000000000000}+00000000	838
Name	{00000093-0000-0000-0000-000000000000}+00000000	147

PMCRUD

The following table describes the property class name, class ID, and long value for the *PMCRUD* object type:

Property Class Name	Class ID	Long
PMActivityRef	{00000314-0000-0000-0000-000000000000}+00000000	788
PMArrowType	{0000031A-0000-0000-0000-000000000000}+00000000	794
PMCreateFlag	{0000033A-0000-0000-0000-000000000000}+00000000	826
PMDeleteFlag	{00000347-0000-0000-0000-000000000000}+00000000	839
PMMergeChangeFlag	{000004B8-0000-0000-0000-000000000000}+00000000	1208

Property Class Name	Class ID	Long
PMReadFlag	{00000394-0000-0000-0000-000000000000}+00000000	916
PMUpdateFlag	{000003C4-0000-0000-0000-000000000000}+00000000	964

PMDataStore

The following table describes the property class name, class ID, and long value for the *PMDataStore* object type:

Property Class Name	Class ID	Long
PMArrowRef	{00000318-0000-0000-0000-000000000000}+00000000	792
PMAuthor	{0000031C-0000-0000-0000-000000000000}+00000000	796
PMBackgroundColorReference	{00000320-0000-0000-0000-000000000000}+00000000	800
PMBoxType	{000004E8-0000-0000-0000-000000000000}+00000000	1256
PMConstraintsString	{00000333-0000-0000-0000-000000000000}+00000000	819
PMDatastoreNumber	{0000033C-0000-0000-0000-000000000000}+00000000	828
PMDefinitionString	{00000346-0000-0000-0000-000000000000}+00000000	838
PMDescriptionString	{00000348-0000-0000-0000-000000000000}+00000000	840
PMEntityRef	{0000035D-0000-0000-0000-000000000000}+00000000	861
PMFacts	{00000361-0000-0000-0000-000000000000}+00000000	865
PMNote	{0000037E-0000-0000-0000-000000000000}+00000000	894
PMObjectsString	{00000384-0000-0000-0000-000000000000}+00000000	900
PMRRGAssocRefList	{000004E0-0000-0000-0000-000000000000}+00000000	1248
PMSource	{000003AF-0000-0000-0000-000000000000}+00000000	943
PMStatusString	{000003B3-0000-0000-0000-000000000000}+00000000	947
PMTextColorReference	{000003BA-0000-0000-0000-000000000000}+00000000	954
PMTitleColorReference	{000003BF-0000-0000-0000-000000000000}+00000000	959
Name	{00000093-0000-0000-0000-000000000000}+00000000	147

PMDiagram

The following table describes the property class name, class ID, and long value for the *PMDiagram* object type:

Property Class Name	Class ID	Long
PMActiveAreaDimension	{00000312-0000-0000-0000-000000000000}+00000000	786
PMArcRadius	{00000315-0000-0000-0000-000000000000}+00000000	789
PMBitmapJustification	{000004CB-0000-0000-0000-000000000000}+00000000	1227
PMBorderSystemDefaultFontRef	{00000323-0000-0000-0000-000000000000}+00000000	803
PMBorderUserFontRef	{00000325-0000-0000-0000-000000000000}+00000000	805
PMBottomRef	{00000326-0000-0000-0000-000000000000}+00000000	806
PMCNumber	{0000032C-0000-0000-0000-000000000000}+00000000	812
PMCreatedDate	{0000033B-0000-0000-0000-000000000000}+00000000	827
PMCurrentDiagram	{00000523-0000-0000-0000-000000000000}+00000000	1315
PMDecompositionNumber	{00000341-0000-0000-0000-000000000000}+00000000	833
PMDecompositionOfRef	{00000342-0000-0000-0000-000000000000}+00000000	834
PMDefaultActivityFontRef	{00000344-0000-0000-0000-000000000000}+00000000	836
PMDefaultArrowFontRef	{00000345-0000-0000-0000-000000000000}+00000000	837
PMDiagramNumber	{0000034A-0000-0000-0000-000000000000}+00000000	842
PMDimension	{0000034E-0000-0000-0000-000000000000}+00000000	846
PMDisplayOffset	{000004B4-0000-0000-0000-000000000000}+00000000	1204
PMDrawStyle	{000004CA-0000-0000-0000-000000000000}+00000000	1226
PMFEOFlag	{00000360-0000-0000-0000-000000000000}+00000000	864
PMHdrFtrType0	{00000519-0000-0000-0000-000000000000}+00000000	1305
PMHdrFtrType1	{0000051A-0000-0000-0000-000000000000}+00000000	1306
PMHdrFtrType2	{0000051B-0000-0000-0000-000000000000}+00000000	1307
PMHdrFtrType3	{0000051C-0000-0000-0000-000000000000}+00000000	1308
PMHdrFtrType4	{0000051D-0000-0000-0000-000000000000}+00000000	1309
PMHdrFtrType5	{0000051E-0000-0000-0000-000000000000}+00000000	1310
PMLeftRef	{0000036D-0000-0000-0000-000000000000}+00000000	877
PMMethodologyCode	{00000372-0000-0000-0000-000000000000}+00000000	882

Property Class Name	Class ID	Long
PMOriginOffset	{00000387-0000-0000-0000-000000000000}+00000000	903
PMPageNumberString	{0000038B-0000-0000-0000-000000000000}+00000000	907
PMParentDiagramRef	{00000427-0000-0000-0000-000000000000}+00000000	1063
PMRepresentedActivityRef	{00000397-0000-0000-0000-000000000000}+00000000	919
PMRightRef	{00000398-0000-0000-0000-000000000000}+00000000	920
PMScenario	{0000039B-0000-0000-0000-000000000000}+00000000	923
PMShowIDEF0KitFlag	{000003A8-0000-0000-0000-000000000000}+00000000	936
PMShowIDEF0TitleFlag	{000003AA-0000-0000-0000-000000000000}+00000000	938
PMStateFlags	{000004E2-0000-0000-0000-000000000000}+00000000	1250
PMStatusString	{000003B3-0000-0000-0000-000000000000}+00000000	947
PMSwimLaneRoleRef	{000004C7-0000-0000-0000-000000000000}+00000000	1223
PMSwimLaneRoleType	{000004E3-0000-0000-0000-000000000000}+00000000	1251
PMSwimLaneUDPRef	{000004C6-0000-0000-0000-000000000000}+00000000	1222
PMSystemLastRevisionDate	{000003B5-0000-0000-0000-000000000000}+00000000	949
PMTextBlockFontRef	{000003B8-0000-0000-0000-000000000000}+00000000	952
PMTextFontRef	{000003BB-0000-0000-0000-000000000000}+00000000	955
PMTextString	{000003BC-0000-0000-0000-000000000000}+00000000	956
PMTitleFontRef	{000003C0-0000-0000-0000-000000000000}+00000000	960
PMTopRef	{000003C2-0000-0000-0000-000000000000}+00000000	962
PMTxtHgtToBmpHgt	{000004E1-0000-0000-0000-000000000000}+00000000	1249
PMUsedAtString	{000003C5-0000-0000-0000-000000000000}+00000000	965
PMUserLastRevisionDate	{000003C6-0000-0000-0000-000000000000}+00000000	966
PMVersion	{00000425-0000-0000-0000-000000000000}+00000000	1061
PMZoomScale	{000004B2-0000-0000-0000-000000000000}+00000000	1202
Name	{00000093-0000-0000-0000-000000000000}+00000000	147

PMERwinAttribute

The following table describes the property class name, class ID, and long value for the *PMERwinAttribute* object type:

Property Class Name	Class ID	Long
PMBPwinGeneratedFlag	{0000031F-0000-0000-0000-000000000000}+00000000	799
PMBPWinOnlyFlag	{0000031E-0000-0000-0000-000000000000}+00000000	798
PMDefinitionString	{00000346-0000-0000-0000-000000000000}+00000000	838
PMERAttributeId	{00000358-0000-0000-0000-000000000000}+00000000	856
PMERDataType	{000004E4-0000-0000-0000-000000000000}+00000000	1252
PMParentname	{00000321-0000-0000-0000-000000000000}+00000000	801
Name	{00000093-0000-0000-0000-000000000000}+00000000	147

PMERwinEntity

The following table describes the property class name, class ID, and long value for the *PMERwinEntity* object type:

Property Class Name	Class ID	Long
PMBPwinGeneratedFlag	{0000031F-0000-0000-0000-000000000000}+00000000	799
PMBPWinOnlyFlag	{0000031E-0000-0000-0000-000000000000}+00000000	798
PMDefinitionString	{00000346-0000-0000-0000-000000000000}+00000000	838
PMEREntityId	{0000035A-0000-0000-0000-000000000000}+00000000	858
PMERModelRef	{00000517-0000-0000-0000-000000000000}+00000000	1303
Name	{00000093-0000-0000-0000-000000000000}+00000000	147

PMERwinModel

The following table describes the property class name, class ID, and long value for the *PMERwinModel* object type:

Property Class Name	Class ID	Long
PMERModelMMId	{00000520-0000-0000-0000-000000000000}+00000000	1312
Name	{00000093-0000-0000-0000-000000000000}+00000000	147

PMExternal

The following table describes the property class name, class ID, and long value for the *PMExternal* object type:

Property Class Name	Class ID	Long
PMArrowRef	{00000318-0000-0000-0000-000000000000}+00000000	792
PMAuthor	{0000031C-0000-0000-0000-000000000000}+00000000	796
PMBackgroundColorReference	{00000320-0000-0000-0000-000000000000}+00000000	800
PMBoxType	{000004E8-0000-0000-0000-000000000000}+00000000	1256
PMConstraintsString	{00000333-0000-0000-0000-000000000000}+00000000	819
PMDefinitionString	{00000346-0000-0000-0000-000000000000}+00000000	838
PMDescriptionString	{00000348-0000-0000-0000-000000000000}+00000000	840
PMEntityRef	{0000035D-0000-0000-0000-000000000000}+00000000	861
PMExternalNumber	{0000035E-0000-0000-0000-000000000000}+00000000	862
PMFacts	{00000361-0000-0000-0000-000000000000}+00000000	865
PMNote	{0000037E-0000-0000-0000-000000000000}+00000000	894
PMObjectsString	{00000384-0000-0000-0000-000000000000}+00000000	900
PMRRGAssocRefList	{000004E0-0000-0000-0000-000000000000}+00000000	1248
PMSource	{000003AF-0000-0000-0000-000000000000}+00000000	943
PMStatusString	{000003B3-0000-0000-0000-000000000000}+00000000	947
PMTextColorReference	{000003BA-0000-0000-0000-000000000000}+00000000	954
PMTitleColorReference	{000003BF-0000-0000-0000-000000000000}+00000000	959
Name	{00000093-0000-0000-0000-000000000000}+00000000	147

PMFont

The following table describes the property class name, class ID, and long value for the *PMFont* object type:

Property Class Name	Class ID	Long
PMFontSize	{00000489-0000-0000-0000-000000000000}+00000000	1161
PMLogfont	{0000036E-0000-0000-0000-000000000000}+00000000	878

PMIRUN

The following table describes the property class name, class ID, and long value for the *PMIRUN* object type:

Property Class Name	Class ID	Long
PMActivityRef	{00000314-0000-0000-0000-000000000000}+00000000	788
PMArrowType	{0000031A-0000-0000-0000-000000000000}+00000000	794
PMInsertFlag	{00000367-0000-0000-0000-000000000000}+00000000	871
PMMergeChangeFlag	{000004B8-0000-0000-0000-000000000000}+00000000	1208
PMNullFlag	{0000037F-0000-0000-0000-000000000000}+00000000	895
PMReadFlag	{00000394-0000-0000-0000-000000000000}+00000000	916
PMUpdateFlag	{000003C4-0000-0000-0000-000000000000}+00000000	964

PMJunction

The following table describes the property class name, class ID, and long value for the *PMJunction* object type:

Property Class Name	Class ID	Long
PMAuthor	{0000031C-0000-0000-0000-000000000000}+00000000	796
PMBackgroundColorReference	{00000320-0000-0000-0000-000000000000}+00000000	800
PMBoxType	{000004E8-0000-0000-0000-000000000000}+00000000	1256
PMConstraintsString	{00000333-0000-0000-0000-000000000000}+00000000	819
PMDefinitionString	{00000346-0000-0000-0000-000000000000}+00000000	838
PMDescriptionString	{00000348-0000-0000-0000-000000000000}+00000000	840
PMFacts	{00000361-0000-0000-0000-000000000000}+00000000	865
PMJunctionBoxNumber	{0000041F-0000-0000-0000-000000000000}+00000000	1055
PMJunctionBoxTypeString	{00000409-0000-0000-0000-000000000000}+00000000	1033
PMNote	{0000037E-0000-0000-0000-000000000000}+00000000	894
PMObjectsString	{00000384-0000-0000-0000-000000000000}+00000000	900
PMRRGAssocRefList	{000004E0-0000-0000-0000-000000000000}+00000000	1248
PMSource	{000003AF-0000-0000-0000-000000000000}+00000000	943
PMStatusString	{000003B3-0000-0000-0000-000000000000}+00000000	947

Property Class Name	Class ID	Long
PMTextColorReference	{000003BA-0000-0000-0000-000000000000}+00000000	954
PMTitleColorReference	{000003BF-0000-0000-0000-000000000000}+00000000	959
Name	{00000093-0000-0000-0000-000000000000}+00000000	147

PMModel

The following table describes the property class name, class ID, and long value for the *PMModel* object type:

Property Class Name	Class ID	Long
PMActBoxCornerDisplayType	{000004A8-0000-0000-0000-000000000000}+00000000	1192
PMActiveAreaDimension	{00000312-0000-0000-0000-000000000000}+00000000	786
PMActivityNumberConventionFlag	{00000313-0000-0000-0000-000000000000}+00000000	787
PMAllowBoxMovementFlag	{000004A4-0000-0000-0000-000000000000}+00000000	1188
PMAllowBoxResizeFlag	{000004A5-0000-0000-0000-000000000000}+00000000	1189
PMArrowNameColorReference	{00000317-0000-0000-0000-000000000000}+00000000	791
PMArrowTextColorReference	{00000319-0000-0000-0000-000000000000}+00000000	793
PMAuthor	{0000031C-0000-0000-0000-000000000000}+00000000	796
PMAuthorsInitials	{0000031D-0000-0000-0000-000000000000}+00000000	797
PMAutospaceArrowsFlag	{000004A7-0000-0000-0000-000000000000}+00000000	1191
PMBitmapJustification	{000004CB-0000-0000-0000-000000000000}+00000000	1227
PMBitmapRefList	{000004DF-0000-0000-0000-000000000000}+00000000	1247
PMBorderSystemDefaultFontRef	{00000323-0000-0000-0000-000000000000}+00000000	803
PMBorderSystemFontRef	{00000324-0000-0000-0000-000000000000}+00000000	804
PMBorderUserFontRef	{00000325-0000-0000-0000-000000000000}+00000000	805
PMBoxBackgroundColorReference	{00000327-0000-0000-0000-000000000000}+00000000	807
PMBpxPath	{0000032A-0000-0000-0000-000000000000}+00000000	810
PMContextDiagramActivityFontRef	{00000334-0000-0000-0000-000000000000}+00000000	820
PMContextDiagramArrowFontRef	{00000335-0000-0000-0000-000000000000}+00000000	821
PMCostDecimalPlaces	{00000338-0000-0000-0000-000000000000}+00000000	824

Property Class Name	Class ID	Long
PMCostUnit	{00000339-0000-0000-0000-000000000000}+00000000	825
PMCreatedDate	{0000033B-0000-0000-0000-000000000000}+00000000	827
PMCreationBuild	{00000522-0000-0000-0000-000000000000}+00000000	1314
PMCreationDataRev	{000004AC-0000-0000-0000-000000000000}+00000000	1196
PMCurrencyPlacement	{000004BA-0000-0000-0000-000000000000}+00000000	1210
PMCurrencySymbol	{000004B9-0000-0000-0000-000000000000}+00000000	1209
PMDataUsageRules	{000004AE-0000-0000-0000-000000000000}+00000000	1198
PMDecompDiagramActivityFontRef	{0000033F-0000-0000-0000-000000000000}+00000000	831
PMDecompDiagramArrowFontRef	{00000340-0000-0000-0000-000000000000}+00000000	832
PMDescriptionString	{00000348-0000-0000-0000-000000000000}+00000000	840
PMDiagramFontRef	{00000349-0000-0000-0000-000000000000}+00000000	841
PMDiagramNumberConventionFlag	{0000034B-0000-0000-0000-000000000000}+00000000	843
PMDiagramTitleColorReference	{0000034C-0000-0000-0000-000000000000}+00000000	844
PMDiagramTitleFontRef	{0000034D-0000-0000-0000-000000000000}+00000000	845
PMDictSettingStringsRef	{000004EA-0000-0000-0000-000000000000}+00000000	1258
PMDimension	{0000034E-0000-0000-0000-000000000000}+00000000	846
PMDisplayBoxHighlightingFlag	{000004AA-0000-0000-0000-000000000000}+00000000	1194
PMDisplayColorsFlag	{000004A0-0000-0000-0000-000000000000}+00000000	1184
PMDisplayLeafCornersFlag	{000004A2-0000-0000-0000-000000000000}+00000000	1186
PMDisplayShadowsFlag	{000004A1-0000-0000-0000-000000000000}+00000000	1185
PMDisplaySquigglesFlag	{000004A3-0000-0000-0000-000000000000}+00000000	1187
PMDrawStyle	{000004CA-0000-0000-0000-000000000000}+00000000	1226
PMDurationDecimalPlaces	{00000357-0000-0000-0000-000000000000}+00000000	855
PMERSourceType	{000004FB-0000-0000-0000-000000000000}+00000000	1275
PMFilename	{00000362-0000-0000-0000-000000000000}+00000000	866
PMFitActivityTextInBoxFlag	{000004A6-0000-0000-0000-000000000000}+00000000	1190
PMFrequencyDecimalPlaces	{00000365-0000-0000-0000-000000000000}+00000000	869
PMHdrFtrType0	{00000519-0000-0000-0000-000000000000}+00000000	1305

Property Class Name	Class ID	Long
PMHdrFtrType1	{0000051A-0000-0000-0000-000000000000}+00000000	1306
PMHdrFtrType2	{0000051B-0000-0000-0000-000000000000}+00000000	1307
PMHdrFtrType3	{0000051C-0000-0000-0000-000000000000}+00000000	1308
PMHdrFtrType4	{0000051D-0000-0000-0000-000000000000}+00000000	1309
PMHdrFtrType5	{0000051E-0000-0000-0000-000000000000}+00000000	1310
PMLongDateFlag	{00000370-0000-0000-0000-000000000000}+00000000	880
PMMergeControlFlag	{000004B5-0000-0000-0000-000000000000}+00000000	1205
PMMergeMasterModelFlag	{000004B6-0000-0000-0000-000000000000}+00000000	1206
PMMergeSubModelFlag	{000004B7-0000-0000-0000-000000000000}+00000000	1207
PMMethodologyCode	{00000372-0000-0000-0000-000000000000}+00000000	882
PMNextBoxNumber	{00000375-0000-0000-0000-000000000000}+00000000	885
PMNextDatastoreNumber	{00000376-0000-0000-0000-000000000000}+00000000	886
PMNextExternalNumber	{00000377-0000-0000-0000-000000000000}+00000000	887
PMNextJunctionNumber	{00000378-0000-0000-0000-000000000000}+00000000	888
PMNoArrowNameMergeFlag	{00000379-0000-0000-0000-000000000000}+00000000	889
PMNoAutoRenumberFlag	{0000037A-0000-0000-0000-000000000000}+00000000	890
PMNoDatastoreNameMergeFlag	{0000037B-0000-0000-0000-000000000000}+00000000	891
PMNodeTreeFontRef	{0000037D-0000-0000-0000-000000000000}+00000000	893
PMNoExternalNameMergeFlag	{0000037C-0000-0000-0000-000000000000}+00000000	892
PMNote	{0000037E-0000-0000-0000-000000000000}+00000000	894
PMOffpageRefIdFlag	{00000386-0000-0000-0000-000000000000}+00000000	902
PMOrgDrawStyle	{000004F7-0000-0000-0000-000000000000}+00000000	1271
PMOriginOffset	{00000387-0000-0000-0000-000000000000}+00000000	903
PMPrefix	{00000390-0000-0000-0000-000000000000}+00000000	912
PMProject	{00000392-0000-0000-0000-000000000000}+00000000	914
PMPurpose	{00000393-0000-0000-0000-000000000000}+00000000	915
PMReportFontRef	{00000395-0000-0000-0000-000000000000}+00000000	917
PMReportMode	{00000396-0000-0000-0000-000000000000}+00000000	918
PMRootActivityNumber	{0000039A-0000-0000-0000-000000000000}+00000000	922
PMScope	{0000039C-0000-0000-0000-000000000000}+00000000	924

Property Class Name	Class ID	Long
PMShapeIdList	{000004DE-0000-0000-0000-000000000000}+00000000	1246
PMShowActivityCostFlag	{000003A0-0000-0000-0000-000000000000}+00000000	928
PMShowActivityNumberingFlag	{000003A1-0000-0000-0000-000000000000}+00000000	929
PMShowActivityPrefixFlag	{000003A2-0000-0000-0000-000000000000}+00000000	930
PMShowArrowBreakFlag	{000003A3-0000-0000-0000-000000000000}+00000000	931
PMShowArrowNameFlag	{000003A4-0000-0000-0000-000000000000}+00000000	932
PMShowDatastoreNumberingFlag	{000003CD-0000-0000-0000-000000000000}+00000000	973
PMShowDatastorePrefixFlag	{000003CF-0000-0000-0000-000000000000}+00000000	975
PMShowDiagramPrefixFlag	{000003A5-0000-0000-0000-000000000000}+00000000	933
PMShowExternalNumberingFlag	{000003CE-0000-0000-0000-000000000000}+00000000	974
PMShowExternalPrefixFlag	{000003D0-0000-0000-0000-000000000000}+00000000	976
PMShowICOMCodeFlag	{000003A6-0000-0000-0000-000000000000}+00000000	934
PMShowIDEF0KitFlag	{000003A8-0000-0000-0000-000000000000}+00000000	936
PMShowIDEF0TitleFlag	{000003AA-0000-0000-0000-000000000000}+00000000	938
PMShowTunnelFlag	{000003AB-0000-0000-0000-000000000000}+00000000	939
PMSortArrowsFlag	{000004AD-0000-0000-0000-000000000000}+00000000	1197
PMSource	{000003AF-0000-0000-0000-000000000000}+00000000	943
PMStatusString	{000003B3-0000-0000-0000-000000000000}+00000000	947
PMSwimLaneBKColorRef	{000004C5-0000-0000-0000-000000000000}+00000000	1221
PMSwimLaneLineColorRef	{000004EE-0000-0000-0000-000000000000}+00000000	1262
PMSwimLaneTextColorRef	{000004ED-0000-0000-0000-000000000000}+00000000	1261
PMSwimLaneTextFontRef	{000004EC-0000-0000-0000-000000000000}+00000000	1260
PMSystemLastRevisionDate	{000003B5-0000-0000-0000-000000000000}+00000000	949
PMTextBlockColorReference	{000003B7-0000-0000-0000-000000000000}+00000000	951
PMTextBlockFontRef	{000003B8-0000-0000-0000-000000000000}+00000000	952
PMTextColorReference	{000003BA-0000-0000-0000-000000000000}+00000000	954
PMTimeFrame	{000003BD-0000-0000-0000-000000000000}+00000000	957
PMTimeUnit	{000003BE-0000-0000-0000-000000000000}+00000000	958
PMTxtHgtToBmpHgt	{000004E1-0000-0000-0000-000000000000}+00000000	1249
PMUserLastRevisionDate	{000003C6-0000-0000-0000-000000000000}+00000000	966

Property Class Name	Class ID	Long
PMViewpoint	{000003CA-0000-0000-0000-000000000000}+00000000	970
Name	{00000093-0000-0000-0000-000000000000}+00000000	147

PMNodeTree

The following table describes the property class name, class ID, and long value for the *PMNodeTree* object type:

Property Class Name	Class ID	Long
PMActiveAreaDimension	{00000312-0000-0000-0000-000000000000}+00000000	786
PMArcRadius	{00000315-0000-0000-0000-000000000000}+00000000	789
PMBitmapHeight	{000004CD-0000-0000-0000-000000000000}+00000000	1229
PMBitmapJustification	{000004CB-0000-0000-0000-000000000000}+00000000	1227
PMBorderSystemDefaultFontRef	{00000323-0000-0000-0000-000000000000}+00000000	803
PMBorderUserFontRef	{00000325-0000-0000-0000-000000000000}+00000000	805
PMBoxDisplayFlags	{000004DB-0000-0000-0000-000000000000}+00000000	1243
PMBoxWidthPrefFlag	{00000329-0000-0000-0000-000000000000}+00000000	809
PMBulletLastFlag	{0000032B-0000-0000-0000-000000000000}+00000000	811
PMCNumber	{0000032C-0000-0000-0000-000000000000}+00000000	812
PMCreatedDate	{0000033B-0000-0000-0000-000000000000}+00000000	827
PMCurrentDiagram	{00000523-0000-0000-0000-000000000000}+00000000	1315
PMDecompositionNumber	{00000341-0000-0000-0000-000000000000}+00000000	833
PMDecompositionOfRef	{00000342-0000-0000-0000-000000000000}+00000000	834
PMDefaultActivityFontRef	{00000344-0000-0000-0000-000000000000}+00000000	836
PMDefaultArrowFontRef	{00000345-0000-0000-0000-000000000000}+00000000	837
PMDefaultBoxWidth	{000004CC-0000-0000-0000-000000000000}+00000000	1228
PMDiagramNumber	{0000034A-0000-0000-0000-000000000000}+00000000	842
PMDimension	{0000034E-0000-0000-0000-000000000000}+00000000	846
PMDisplayOffset	{000004B4-0000-0000-0000-000000000000}+00000000	1204
PMDrawBorderFlag	{00000353-0000-0000-0000-000000000000}+00000000	851
PMDrawBoxFlag	{00000354-0000-0000-0000-000000000000}+00000000	852

Property Class Name	Class ID	Long
PMDrawStyle	{000004CA-0000-0000-0000-000000000000}+00000000	1226
PMHdrFtrType0	{00000519-0000-0000-0000-000000000000}+00000000	1305
PMHdrFtrType1	{0000051A-0000-0000-0000-000000000000}+00000000	1306
PMHdrFtrType2	{0000051B-0000-0000-0000-000000000000}+00000000	1307
PMHdrFtrType3	{0000051C-0000-0000-0000-000000000000}+00000000	1308
PMHdrFtrType4	{0000051D-0000-0000-0000-000000000000}+00000000	1309
PMHdrFtrType5	{0000051E-0000-0000-0000-000000000000}+00000000	1310
PMMethodologyCode	{00000372-0000-0000-0000-000000000000}+00000000	882
PMNumberInBoxFlag	{00000381-0000-0000-0000-000000000000}+00000000	897
PMNumLevels	{00000380-0000-0000-0000-000000000000}+00000000	896
PMOriginOffset	{00000387-0000-0000-0000-000000000000}+00000000	903
PMOrthogonalFlag	{000004CE-0000-0000-0000-000000000000}+00000000	1230
PMPageNumberString	{0000038B-0000-0000-0000-000000000000}+00000000	907
PMParentDiagramRef	{00000427-0000-0000-0000-000000000000}+00000000	1063
PMRepresentedActivityRef	{00000397-0000-0000-0000-000000000000}+00000000	919
PMShowIDEF0KitFlag	{000003A8-0000-0000-0000-000000000000}+00000000	936
PMShowIDEF0TitleFlag	{000003AA-0000-0000-0000-000000000000}+00000000	938
PMStateFlags	{000004E2-0000-0000-0000-000000000000}+00000000	1250
PMStatusString	{000003B3-0000-0000-0000-000000000000}+00000000	947
PMSwimLaneRoleRef	{000004C7-0000-0000-0000-000000000000}+00000000	1223
PMSwimLaneRoleType	{000004E3-0000-0000-0000-000000000000}+00000000	1251
PMSwimLaneUDPRef	{000004C6-0000-0000-0000-000000000000}+00000000	1222
PMSystemLastRevisionDate	{000003B5-0000-0000-0000-000000000000}+00000000	949
PMTextBlockFontRef	{000003B8-0000-0000-0000-000000000000}+00000000	952
PMTextFontRef	{000003BB-0000-0000-0000-000000000000}+00000000	955
PMTextString	{000003BC-0000-0000-0000-000000000000}+00000000	956
PMTitleFontRef	{000003C0-0000-0000-0000-000000000000}+00000000	960
PMTxtHgtToBmpHgt	{000004E1-0000-0000-0000-000000000000}+00000000	1249
PMUsedAtString	{000003C5-0000-0000-0000-000000000000}+00000000	965
PMUserLastRevisionDate	{000003C6-0000-0000-0000-000000000000}+00000000	966

Property Class Name	Class ID	Long
PMVersion	{00000425-0000-0000-0000-000000000000}+00000000	1061
PMZoomScale	{000004B2-0000-0000-0000-000000000000}+00000000	1202
Name	{00000093-0000-0000-0000-000000000000}+00000000	147

PMNode

The following table describes the property class name, class ID, and long value for the *PMNode* object type:

Property Class Name	Class ID	Long
PMPointLocation	{0000044F-0000-0000-0000-000000000000}+00000000	1103

PMOrgChartBox

The following table describes the property class name, class ID, and long value for the *PMOrgChartBox* object type:

Property Class Name	Class ID	Long
PMBoxCoordinates	{0000044E-0000-0000-0000-000000000000}+00000000	1102
PMOrgBoxBKColorReference	{000004D9-0000-0000-0000-000000000000}+00000000	1241
PMOrgBoxHasParent	{000004D8-0000-0000-0000-000000000000}+00000000	1240
PMOrgBoxId	{000004D2-0000-0000-0000-000000000000}+00000000	1234
PMOrgBoxParentId	{000004D3-0000-0000-0000-000000000000}+00000000	1235
PMRRGAssocRef	{0000051F-0000-0000-0000-000000000000}+00000000	1311
PMShapeColorReference	{000004DA-0000-0000-0000-000000000000}+00000000	1242
PMTextColorReference	{000003BA-0000-0000-0000-000000000000}+00000000	954

PMOrgChartRow

The following table describes the property class name, class ID, and long value for the *PMOrgChartRow* object type:

Property Class Name	Class ID	Long
PMFirstBoxInRow	{000004CF-0000-0000-0000-000000000000}+00000000	1231
PMMaxRowBoxCount	{000004D1-0000-0000-0000-000000000000}+00000000	1233

Property Class Name	Class ID	Long
PMOrgRowId	{00000521-0000-0000-0000-000000000000}+00000000	1313
PMRowBoxCount	{000004D0-0000-0000-0000-000000000000}+00000000	1232

PMPalette

The following table describes the property class name, class ID, and long value for the *PMPalette* object type:

Property Class Name	Class ID	Long
PMVersion	{00000425-0000-0000-0000-000000000000}+00000000	1061

PMReferent

The following table describes the property class name, class ID, and long value for the *PMReferent* object type:

Property Class Name	Class ID	Long
PMArrowRef	{00000318-0000-0000-0000-000000000000}+00000000	792
PMAuthor	{0000031C-0000-0000-0000-000000000000}+00000000	796
PMBackgroundColorReference	{00000320-0000-0000-0000-000000000000}+00000000	800
PMBoxType	{000004E8-0000-0000-0000-000000000000}+00000000	1256
PMConstraintsString	{00000333-0000-0000-0000-000000000000}+00000000	819
PMDefinitionString	{00000346-0000-0000-0000-000000000000}+00000000	838
PMDescriptionString	{00000348-0000-0000-0000-000000000000}+00000000	840
PMEntityRef	{0000035D-0000-0000-0000-000000000000}+00000000	861
PMFacts	{00000361-0000-0000-0000-000000000000}+00000000	865
PMNote	{0000037E-0000-0000-0000-000000000000}+00000000	894
PMObjectsString	{00000384-0000-0000-0000-000000000000}+00000000	900
PMRRGAssocRefList	{000004E0-0000-0000-0000-000000000000}+00000000	1248
PMSource	{000003AF-0000-0000-0000-000000000000}+00000000	943
PMStatusString	{000003B3-0000-0000-0000-000000000000}+00000000	947
PMTextColorReference	{000003BA-0000-0000-0000-000000000000}+00000000	954

Property Class Name	Class ID	Long
PMTitleColorReference	{000003BF-0000-0000-0000-000000000000}+00000000	959
Name	{00000093-0000-0000-0000-000000000000}+00000000	147

PMResource

The following table describes the property class name, class ID, and long value for the *PMResource* object type:

Property Class Name	Class ID	Long
PMDescriptionStringCopy	{0000036F-0000-0000-0000-000000000000}+00000000	879
NameCopy	{00000094-0000-0000-0000-000000000000}+00000000	148

PMRole

The following table describes the property class name, class ID, and long value for the *PMRole* object type:

Property Class Name	Class ID	Long
PMBitmapRef	{000004C8-0000-0000-0000-000000000000}+00000000	1224
PMDescriptionStringCopy	{0000036F-0000-0000-0000-000000000000}+00000000	879
PMImportance	{000004FE-0000-0000-0000-000000000000}+00000000	1278
PMShapeId	{000004C9-0000-0000-0000-000000000000}+00000000	1225
NameCopy	{00000094-0000-0000-0000-000000000000}+00000000	148

PMRoleGroup

The following table describes the property class name, class ID, and long value for the *PMRoleGroup* object type:

Property Class Name	Class ID	Long
PMBitmapRef	{000004C8-0000-0000-0000-000000000000}+00000000	1224
PMDescriptionStringCopy	{0000036F-0000-0000-0000-000000000000}+00000000	879
PMImportance	{000004FE-0000-0000-0000-000000000000}+00000000	1278
PMShapeId	{000004C9-0000-0000-0000-000000000000}+00000000	1225
NameCopy	{00000094-0000-0000-0000-000000000000}+00000000	148

PMRRGAssociation

The following table describes the property class name, class ID, and long value for the *PMRRGAssociation* object type:

Property Class Name	Class ID	Long
PMResourceRef	{00000518-0000-0000-0000-000000000000}+00000000	1304
PMRoleRef	{000004E6-0000-0000-0000-000000000000}+00000000	1254
PMRoleTypeRef	{00000500-0000-0000-0000-000000000000}+00000000	1280

PMRuler

The following table describes the property class name, class ID, and long value for the *PMRuler* object type:

Property Class Name	Class ID	Long
PMRulerBKColorRef	{0000050E-0000-0000-0000-000000000000}+00000000	1294
PMRulerDisplayFlags	{00000513-0000-0000-0000-000000000000}+00000000	1299
PMRulerLabel	{00000501-0000-0000-0000-000000000000}+00000000	1281
PMRulerLabelColorRef	{00000512-0000-0000-0000-000000000000}+00000000	1298
PMRulerLeftValueDate	{00000505-0000-0000-0000-000000000000}+00000000	1285
PMRulerLeftValueDbl	{00000507-0000-0000-0000-000000000000}+00000000	1287
PMRulerLeftValueLong	{00000503-0000-0000-0000-000000000000}+00000000	1283
PMRulerMajorDivColorRef	{0000050F-0000-0000-0000-000000000000}+00000000	1295
PMRulerMajorIncDbl	{0000050B-0000-0000-0000-000000000000}+00000000	1291
PMRulerMajorIncLong	{00000509-0000-0000-0000-000000000000}+00000000	1289
PMRulerMinorDivColorRef	{00000510-0000-0000-0000-000000000000}+00000000	1296
PMRulerMinorIncDbl	{0000050C-0000-0000-0000-000000000000}+00000000	1292
PMRulerMinorIncLong	{0000050A-0000-0000-0000-000000000000}+00000000	1290
PMRulerRightValueDate	{00000506-0000-0000-0000-000000000000}+00000000	1286
PMRulerRightValueDbl	{00000508-0000-0000-0000-000000000000}+00000000	1288
PMRulerRightValueLong	{00000504-0000-0000-0000-000000000000}+00000000	1284
PMRulerStyle	{0000050D-0000-0000-0000-000000000000}+00000000	1293
PMRulerUnitsColorRef	{00000511-0000-0000-0000-000000000000}+00000000	1297

Property Class Name	Class ID	Long
PMRulerUnitsLabelFontRef	{00000516-0000-0000-0000-000000000000}+00000000	1302
PMRulerUnitsTextFontRef	{00000514-0000-0000-0000-000000000000}+00000000	1300
PMRulerUnitsUDPRef	{00000502-0000-0000-0000-000000000000}+00000000	1282

PMTextBlock

The following table describes the property class name, class ID, and long value for the *PMTextBlock* object type:

Property Class Name	Class ID	Long
PMIsPurposeFlag	{00000369-0000-0000-0000-000000000000}+00000000	873
PMIsViewpointFlag	{0000036A-0000-0000-0000-000000000000}+00000000	874
PMTextBlockColorReference	{000003B7-0000-0000-0000-000000000000}+00000000	951
PMTextBlockFontRef	{000003B8-0000-0000-0000-000000000000}+00000000	952
PMTextBoundingBox	{000003B9-0000-0000-0000-000000000000}+00000000	953
PMTextString	{000003BC-0000-0000-0000-000000000000}+00000000	956

PMTunnel

The following table describes the property class name, class ID, and long value for the *PMTunnel* object type:

Property Class Name	Class ID	Long
PMExternalRef	{0000035F-0000-0000-0000-000000000000}+00000000	863
PMOffpageRef	{00000385-0000-0000-0000-000000000000}+00000000	901

PMUDPCategory

The following table describes the property class name, class ID, and long value for the *PMUDPCategory* object type:

Property Class Name	Class ID	Long
PMDispInActivityDispFlag	{0000034F-0000-0000-0000-000000000000}+00000000	847
PMDispInActivityRptFlag	{00000350-0000-0000-0000-000000000000}+00000000	848
PMDispInArrowDispFlag	{00000351-0000-0000-0000-000000000000}+00000000	849

Property Class Name	Class ID	Long
PMDispInArrowRptFlag	{00000352-0000-0000-0000-000000000000}+00000000	850
PMSequenceNumber	{0000039E-0000-0000-0000-000000000000}+00000000	926
Name	{00000093-0000-0000-0000-000000000000}+00000000	147

PMUDPDefinition

The following table describes the property class name, class ID, and long value for the *PMUDPDefinition* object type:

Property Class Name	Class ID	Long
PMCategories	{0000032D-0000-0000-0000-000000000000}+00000000	813
PMDateStyleFlag	{0000033D-0000-0000-0000-000000000000}+00000000	829
PMPrecision	{0000038F-0000-0000-0000-000000000000}+00000000	911
PMSequenceNumber	{0000039E-0000-0000-0000-000000000000}+00000000	926
PMUDPDescription	{000004C3-0000-0000-0000-000000000000}+00000000	1219
PMUDPMultiSelectFlag	{000004B0-0000-0000-0000-000000000000}+00000000	1200
PMValueType	{000003C8-0000-0000-0000-000000000000}+00000000	968
Name	{00000093-0000-0000-0000-000000000000}+00000000	147

PMUDPInstance

The following table describes the property class name, class ID, and long value for the *PMUDPInstance* object type:

Property Class Name	Class ID	Long
PMDefinedByRef	{000003CB-0000-0000-0000-000000000000}+00000000	971
PMValueOfRef	{000003C7-0000-0000-0000-000000000000}+00000000	967
PMValueOfRefSet	{000004B1-0000-0000-0000-000000000000}+00000000	1201

PMUDPValue

The following table describes the property class name, class ID, and long value for the *PMUDPValue* object type:

Property Class Name	Class ID	Long
PMDateUDPValue	{0000033E-0000-0000-0000-000000000000}+00000000	830
PMFloatUDPValue	{00000363-0000-0000-0000-000000000000}+00000000	867
PMIntUDPValue	{00000368-0000-0000-0000-000000000000}+00000000	872
PMSequenceUDPValue	{0000039F-0000-0000-0000-000000000000}+00000000	927
PMStringUDPValue	{000003B4-0000-0000-0000-000000000000}+00000000	948

Appendix C: Access CA ERwin PM Properties

This appendix describes the CA ERwin PM API property value access. Property value access in the API is handled by the *ISCModelProperty::Value(VARIANT ValueId[optional], long ValueType[optional])* function.

The *ValueId* parameter is used to specify the member of a non-scalar property. If the property is scalar, the *ValueId* parameter is ignored. If the *ValueId* parameter is empty for a non-scalar property, all the values of the property are returned in a SAFEARRAY, provided that the property is homogeneous. For heterogeneous property types such as DT_LPLOGFONT, VT_ERROR is returned.

The *ValueType* parameter is used to indicate the format of the return value. The only valid values for the *ValueType* parameter are SCVT_DEFAULT, SCVT_BSTR (returns the value as a string), and the value type of the property itself.

This section contains the following topics:

Default Return Types of Property Types

The following table shows the default return types of the different property types. Default return values are used when the *ValueId* parameter is not supplied in the *ISCModelProperty::Value* function.

CA ERwin PM Type	Description	VARIANT Type	SCAPI Type
DT_Invalid	Error	VT_ERROR	SCVT_NULL
DT_char	Character (signed)	VT_BSTR	SCVT_BSTR
DT_short	Short	VT_I2	SCVT_I2
DT_long	Long	VT_I4	SCVT_I4
DT_enum	Enum	VT_I2	SCVT_I2
DT_double	Double, float, real	VT_R8	SCVT_R8

CA ERwin PM Type	Description	VARIANT Type	SCAPI Type
DT_LPSTR DT_LPNewStr	Pointer to a string	VT_BSTR	SCVT_BSTR
DT_CString DT_NewCString	Pointer to a CString	VT_BSTR	SCVT_BSTR
DT_LPLOGFONT	Pointer to a LOGFONT	VT_ERROR	SCVT_NULL
DT_BOOL	Boolean value	VT_BOOL	SCVT_BOOLEAN
DT_ref 2	Pointer to a *PMBase* object	VT_BSTR	SCVT_OBJID
DT_LPPMID	Pointer to an OriginId	VT_BSTR	SCVT_OBJID
DT_Byte	Character (unsigned)	VT_UI1	SCVT_UI1
DT_sDate	PM date structure	VT_DATE	SCVT_DATE
DT_POINT	Coordinate point	VT_ARRAY|VT_I4	SCVT_POINT
DT_RECT	Rectangle	VT_ARRAY|VT_I4	SCVT_RECT
DT_sSelList	Multi-select list	VT_ARRAY|VT_I2	SCVT_BLOB
DT_sCharArray	Character Array header	VT_ARRAY|VT_BSTR	SCVT_BLOB
DT_sShortArray	Short Array header	VT_ARRAY|VT_I2	SCVT_BLOB
DT_sLongArray	Long Array header	VT_ARRAY|VT_I4	SCVT_BLOB
DT_sDoubleArray	Double Array header	VT_ARRAY|VT_R8	SCVT_BLOB
DT_sStrArray	String Array header	VT_ARRAY|VT_BSTR	SCVT_BLOB
DT_sByteArray	Byte Array header	VT_ARRAY|VT_UI1	SCVT_BLOB
DT_sPointArray	Coord Array header	VT_ARRAY|VT_I4	SCVT_BLOB
DT_sRefArray	Base Object Ptr Array	VT_ARRAY|VT_BSTR	SCVT_BLOB

Note:

- For CA ERwin PM type DT_LPLOGFONT, the types that are listed as VT_ERROR for the VARIANT Type column cannot be accessed as a scalar value. VT_ERROR is returned if the *ISCModelProperty::Value* function is called without a *ValueId*.
- For CA ERwin PM type DT_sPointArray, the return value is a two-dimensional array where the 0 index in the second dimension is the *x* value of the point, and the 1 index in the second dimension is the *y* value of the point.
- For CA ERwin PM type DT_sRefArray, the object IDs of the references are returned.

Non-scalar Access

To access a specific value within a non-scalar property, the *ValueId* parameter must be supplied in the *ISCModelProperty::Value(VARIANT ValueId [optional], long ValueType [optional])* function. If the property is a structure (such as DT_LPLOGFONT, DT_DATE, and so on) the name of the structure member can be used for the *ValueId* argument. Also, an integer value that maps to the structure member can be used. For arrays, only the index of the array can be used for *ValueId*.

CA ERwin PM Type	Description	VT_BSTR ValueId Value (Member Name)	VT_I2 ValueId Value (Index)	Resulting VARIANT Type	Resulting API Type
DT_LPLOGFONT	Pointer to a LOGFONT	IfHeight	0	VT_I4	SCVT_I4
		IfWidth	1	VT_I4	SCVT_I4
		IfEscapement	2	VT_I4	SCVT_I4
		IfOrientation	3	VT_I4	SCVT_I4
		IfWeight	4	VT_I4	SCVT_I4
		IfItalic	5	VT_IU1	SCVT_UI1
		IfUnderline	6	VT_IU1	SCVT_UI1
		IfStrikeOut	7	VT_IU1	SCVT_UI1
		IfCharSet	8	VT_IU1	SCVT_UI1
		IfOutPrecision	9	VT_IU1	SCVT_UI1
		IfClipPrecision	10	VT_IU1	SCVT_UI1
		IfQuality	11	VT_IU1	SCVT_UI1
		IfPitchAndFamily	12	VT_IU1	SCVT_UI1
		IfFaceName	13	VT_BSTR	SCVT_BSTR
DT_DATE	Date structure	y or year	0	VT_I2	SCVT_I2
		m or month	1	VT_I2	SCVT_I2
		d or day	2	VT_I2	SCVT_I2
DT_POINT	Coordinate point	x	0	VT_I4	SCVT_I4
		y	1	VT_I4	SCVT_I4
DT_RECT	Rectangle	left	0	VT_I4	SCVT_I4
		top	1	VT_I4	SCVT_I4

CA ERwin PM Type	Description	VT_BSTR ValueId Value (Member Name)	VT_I2 ValueId Value (Index)	Resulting VARIANT Type	Resulting API Type
		right	2	VT_I4	SCVT_I4
		bottom	3	VT_I4	SCVT_I4
DT_sSelList	Multi-select list	None	0-based list index	VT_I2	SCVT_I2
DT_sCharArray	Character Array header	None	0-based array index	VT_BSTR	SCVT_BSTR
DT_sShortArray	Short Array header	None	0-based array index	VT_I2	SCVT_I2
DT_sLongArray	Long Array header	None	0-based array index	VT_I4	SCVT_I4
DT_sDoubleArray	Double Array header	None	0-based array index	VT_R8	SCVT_R8
DT_sStrArray	String Array header	None	0-based array index	VT_BSTR	SCVT_BSTR
DT_sByteArray	Byte Array header	None	0-based array index	VT_UI1	SCVT_UI1
DT_sPointArray	Coord Array header	None	0-based array index	VT_ARRAY|VT_I4	SCVT_BLOB
DT_sRefArray	Base Object Ptr Array	None	0-based array index	VT_BSTR	SCVT_OBJID

If there is a valid VT_BSTR for the *ValueId* parameter, the string is case sensitive.

For the array property types, if the upper bounds of the array are exceeded, then VT_ERROR is returned.

Index

C

D

E

F

I

L

M

N

O

P

R

S

T

U

V